Titre: Title:	Méthodes de pénalisation pour l'optimisation de structures
Auteur: Author:	Pierre-Rémi Curatolo
Date:	2008
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Curatolo, P.-R. (2008). Méthodes de pénalisation pour l'optimisation de structures [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8329/

UNIVERSITÉ DE MONTRÉAL

MÉTHODES DE PÉNALISATION POUR L'OPTIMISATION DE STRUCTURES

PIERRE-RÉMI CURATOLO
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
JUILLET 2008

Your file Votre référence
ISBN: 978-0-494-46043-6
Our file Notre référence
ISBN: 978-0-494-46043-6

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

MÉTHODES DE PÉNALISATION POUR L'OPTIMISATION DE STRUCTURES

présenté par: CURATOLO Pierre-Rémi
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. AUDET Charles, Ph.D., président
M. ORBAN Dominique, Doct. Sc., membre et directeur de recherche
M. DUSSAULT Jean-Pierre, Ph.D., membre

Remerciements

Je souhaite remercier les personnes qui m'ont apporté l'énergie, les connaissances et les encouragements nécessaires à la conduite optimale de mes travaux de recherche : Dominique Orban, pour son aide de tous les instants et ses conseils toujours avisés ainsi que Gabrielle et mes parents, pour leur soutien moral.

Résumé

Dans ce document, on aborde le problème de *conception de structure* qui peut se formuler comme un programme mathématique non-linéaire avec contraintes d'égalité et d'inégalité. Ce problème présente une forme de dégénérescence. Pour le résoudre, on utilise une méthode de points intérieurs avec pénalisation ℓ_1. Le terme de pénalité est reformulé, par le biais de variables additionnelles appelées *variables élastiques*, ce qui permet de régulariser le problème. Nous discutons des propriétés des programmes mathématiques associés aux problèmes de structure ainsi que de la capacité théorique de la méthode à trouver des solutions pour ces problèmes. Nous détaillons l'implémentation de l'algorithme et nous présentons aussi les diverses options explorées en vue d'améliorer la méthode d'un point de vue pratique. Des résultats numériques sont fournis pour les collections tests Hock-Schittkowski et MacMPEC ainsi que pour des problèmes de structure de petite taille. L'aspect visualisation des structures obtenues par l'algorithme est également abordé.

Abstract

In this document we tackle the *Truss Design Problem* which can be formulated as a nonlinear mathematical program with both equality and inequality constraints. The problem exhibits a certain type of degeneracy. To solve this problem, we use an interior point ℓ_1-penalty method. We reformulate the penalty term, adding variables called *elastic variables*, which leads to a regular problem. We discuss the mathematical properties of Truss Design Problem models as well as the theorectical ability of our method to identify a solution. We also explain the various numerical enhancements that we implemented to improve the method. Numerical results are provided. Experiments on the Hock-Schittkowski and MacMPEC test collections are carried out. We've also solved small scale Truss Design Problems. We briefly show an approach to visualize the trusses proposed by the algorithm.

Table des matières

Liste des tableaux

Liste des figures

Introduction

L'optimisation de structure se rencontre essentiellement dans le domaine de la construction et s'applique par exemple aux ponts. Un problème classique consiste à considérer un treillis, c'est-à-dire un ensemble de barres reliées entre elles par des noeuds, soumis à des forces extérieures. On cherche alors par exemple à trouver la structure qui possède un volume minimum tout en vérifiant des contraintes qui définissent la résistance du treillis (étant donnés le matériau, la forme géométrique des barres,...) face aux charges extérieures. Pour cela on peut agir sur plusieurs variables, l'une d'entre elles étant le *diamètre des barres*. Ce type de problème a déjà été l'objet de nombreuses études, on citera parmi elles (Achtziger, 1999) ou encore (Kocvara & Outrata, 2006). Une caractéristique essentielle de ces problèmes est qu'ils sont difficiles à résoudre d'un point de vue numérique.

La difficulté réside en bonne partie dans le fait que le domaine admissible pour ce type de problèmes est non-convexe et ne possède pas d'intérieur strict dans certaines régions. Pour cette raison, on a décidé d'approcher ces problèmes à l'aide d'une combinaison des méthodes de points intérieurs, de fonction de mérite et de régularisation. Le principe de cette méthode est d'effectuer une transformation du problème initial afin d'obtenir un problème annexe régulier, appelé *problème élastique*, dont le domaine admissible est plus étendu que celui du problème original. En résolvant une série de problèmes élastiques on est alors en mesure, sous certaines hypothèses, de détecter des points optimaux au premier ordre du problème initial. Cet algorithme nommé PSUPERB semble donc avoir le potentiel pour s'attaquer efficacement aux problèmes de structure.

D'un point de vue théorique, nous avons montré que l'algorithme est capable d'identifier des points optimaux pour lesquels les conditions de qualifications classiques sont violées. Nous avons aussi adapté des résultats sur la convergence de l'algorithme à certaines classes de programmes mathématiques. Numériquement, le premier défi à surmonter a été de faire fonctionner l'algorithme. Ensuite, une part de nos travaux a porté sur la gestion des nombreux paramètres en jeu dans PSUPERB, ce qui s'est avéré être primordial dans le but d'améliorer les performances du code. D'autre part, nous avons essayé de mettre en place un préconditionneur afin

d'augmenter la qualité de l'implémentation. Finalement, nous avons eu l'occasion d'effectuer de nombreux tests numériques sur différentes classes de problèmes. Sachant que les problèmes de structure sont dégénérés d'après (Achtziger, 1999), nous avons d'abord validé PSUPERB sur des problèmes qui se comportent bien. Ensuite, nous sommes passés à des problèmes dégénérés dont la structure a été étudiée auparavant, les MPECs car nous savions que les problèmes de structure ont une modélisation MPEC, mais celle-ci provient en réalité de problèmes bi-niveau (Kocvara & Outrata, 2006). Durant la maîtrise, une autre formulation, nommée MPVC, a été donnée et analysée (Achtziger & Kanzow, 2008). Nous avons donc étendu l'analyse de PSUPERB à celle-ci et validé l'algorithme sur les quelques problèmes disponibles. Nous fournirons dans ce document les analyses et les résultats numériques associés à ces travaux.

Dans un premier temps, nous proposons une partie théorique qui nous permettra de passer en revue les connaissances de programmation mathématique non-linéaire indispensables. Nous aborderons aussi la description précise des problèmes de structure qui nous ont interessés ainsi qu'une présentation détaillée de l'algorithme que nous avons choisi pour résoudre ces problèmes. Dans une deuxième partie, nous nous focaliserons sur l'aspect pratique de la résolution de problèmes de structure avec l'implémentation de l'algorithme ainsi que les diverses stratégies d'amélioration des performances numériques. Enfin, nous présenterons les résultats numériques obtenus sur des collections de problèmes test et sur certains modèles de problème de structure de petite taille.

Chapitre 1

Cadre de travail : problèmes de structure

Dans ce document, on s'intéresse à des programmes mathématiques. On peut définir ces derniers comme étant des problèmes qui se composent d'un objectif, qu'on cherche à minimiser ou maximiser et d'un ensemble de contraintes qu'on doit respecter. Les variables du problèmes sont contenues dans un ensemble Ω. Si on note x les variables, on peut alors écrire

$$\begin{aligned} \underset{x\in\Omega}{\text{minimiser}} \quad & \text{Objectif}(x) \\ \text{s.c. :} \quad & \text{Contraintes}(x). \end{aligned}$$

Remarquons que minimiser l'objectif revient à maximiser l'opposé de ce dernier. Dans la suite, nous choisirons de minimiser les objectifs.

Exemple
Regardons un exemple concret de programme mathématique. Soit le problème

$$\begin{aligned} \underset{x\in\mathbb{R}}{\text{minimiser}} \quad & x^2 \\ \text{s.c. :} \quad & x \geq 2. \end{aligned}$$

Dans cet exemple, x^2 est l'objectif et $x \geq 2$ est la contrainte qu'il faut respecter. L'ensemble Ω correspond à $\mathbb{R}$. Dans ces conditions, la solution est évidemment atteinte en $x = 2$ et cela donne la valeur 4 pour l'objectif.

1.1 Rappels de programmation non-linéaire

Nous allons procéder à de brefs rappels des principes fondamentaux de l'optimisation non-linéaire puisque nous serons amenés à exploiter cette dernière dans toute

la suite de ce document.

1.1.1 Optimisation sans contrainte

Soit $f : \mathbb{R}^n \to \mathbb{R}$ et considérons le problème

$$\underset{x \in \mathbb{R}^n}{\text{minimiser}} \quad f(x). \tag{1.1}$$

Notation 1.1.1.1 *Soit $x \in \mathbb{R}^n$. On appelle boule ouverte de centre x et de rayon ρ, notée $\mathcal{B}(x, \rho)$, l'ensemble des points $\tilde{x} \in \mathbb{R}^n$ tel que $||x - \tilde{x}|| < \rho$.*

Définition 1.1.1.1 *Considérons le problème (1.1) et soit $x^* \in \mathbb{R}^n$. On dit que x^* est un*
(a) minimum global sans contrainte de f ssi $\forall x \in \mathbb{R}^n, f(x) \geq f(x^*)$,
(b) minimum local sans contrainte de f ssi $\exists \epsilon > 0$ tel que $\forall x \in \mathcal{B}(x^*, \epsilon), f(x) \geq f(x^*)$,
(c) minimum local strict de f ssi $\exists \epsilon > 0$ tel que $\forall x \in \mathcal{B}(x^*, \epsilon), x \neq x^*, f(x) > f(x^*)$.

Lorsqu'on a affaire à un problème d'optimisation sans contrainte, on doit se poser les questions suivantes :

(a) existe-t-il une solution ?

(b) si oui, est-elle unique ou en existe-t-il plusieurs ? Les solutions trouvées sont-elles locales ou globales ?

En règle générale, lorsqu'on essaie de résoudre des problèmes non-convexes de grande taille, il est très difficile de trouver des minima globaux et on est donc satisfait lorsqu'on trouve des minima locaux. Il convient à présent de proposer les conditions nécessaires d'optimalité.

Définition 1.1.1.2 *Une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est dite de classe $\mathcal{C}^k$ ssi toutes ses dérivées partielles $\partial f^k / \partial x_{i_1} \partial x_{i_2} ... \partial x_{i_k}$ existent et sont continues, avec $i_1, i_2, ..., i_k$ des entiers compris entre 1 et n.*

Notation 1.1.1.2 *Soit $f : \mathbb{R}^n \to \mathbb{R}$. On note $\nabla f(\tilde{x})$ le gradient et $\nabla^2 f(\tilde{x})$ la matrice Hessienne de f évalués au point $\tilde{x}$.*

Théorème 1.1.1 *(Conditions nécessaires d'optimalité du premier ordre). Soit* $x^* \in \mathbb{R}^n$ *un minimum local de (1.1) et supposons que* f *soit* $\mathcal{C}^1$ *dans une boule ouverte de centre* x^*. *Alors* $\nabla f(x^*) = 0$.

Définition 1.1.1.3 *Un point* $x \in \mathbb{R}^n$ *tel que* $\nabla f(x) = 0$ *est appelé un point* critique du premier ordre *ou* stationnaire.

Théorème 1.1.2 *(Conditions nécessaires d'optimalité du second ordre). Soit* $x^* \in \mathbb{R}^n$ *un minimum local de (1.1) et supposons que* f *soit* $\mathcal{C}^2$ *dans une boule ouverte de centre* x^*. *Alors* $\nabla^2 f(x^*)$ *est semi-définie positive.*

On dispose aussi de conditions suffisantes d'optimalité.

Théorème 1.1.3 *(Conditions suffisantes d'optimalité du second ordre). Soient* $x^* \in \mathbb{R}^n$ *et* f *de classe* $\mathcal{C}^2$ *dans une boule ouverte de centre* x^*. *Si* $\nabla f(x^*) = 0$ *et* $\nabla^2 f(x^*)$ *est définie positive alors* x^* *est un minimum local de (1.1).*

Exemple
Soit $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x) = (x_1 - 1)^2 + (x_2 + 2)^2$. L'unique point stationnaire est $x^* = (x_1^*, x_2^*) = (1, -2)$. De plus $\nabla^2 f(x^*) = 2I$ qui est définie positive donc x^* est un minimum local.

Maintenant qu'on est capable de caractériser des minima dans le cas sans contrainte, on peut se demander ce qui arrive dans le cas où il y a des contraintes.

1.1.2 Optimisation avec contraintes

On considère le programme mathématique suivant :

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimiser}} & f(x) \\ \text{s.c. :} & h(x) = 0, \\ & g(x) \geq 0, \end{array} \tag{NLP}$$

où $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^p$, $g : \mathbb{R}^n \to \mathbb{R}^m$ sont des fonctions de classe $\mathcal{C}^1$.

Notation 1.1.2.1 *On appelle* domaine admissible de (NLP) *et on note* Ω *l'ensemble des points de* $\mathbb{R}^n$ *tel que les contraintes sont respectées, i.e.,*

$$\Omega := \{x \in \mathbb{R}^n \mid h(x) = 0 \text{ et } g(x) \geq 0\}.$$

Dans le cas de l'optimisation sans contrainte on a réussi à caractériser les solutions locales du problème. Le but est maintenant d'essayer d'en faire de même pour le cas de l'optimisation avec contraintes. Puisque les solutions dépendent des contraintes, il convient de s'intéresser à la représentation algébrique du domaine admissible Ω. On parle alors de conditions de qualification.

1.1.2.1 Conditions de qualification

Pour un programme mathématique avec contraintes, les conditions de qualification concernent la représentation algébrique des contraintes en un point donné de Ω.

Définition 1.1.2.1 *(Ensemble actif)*
On dit que la contrainte g_j (pour $j \in 1,...,\ m$) est active en $x^* \in \Omega$ si $g_j(x^*) = 0$. On note $\mathcal{A}(x^*) = \{j \in 1,...,m \mid g_j(x^*) = 0\}$ l'ensemble des contraintes actives en x^*.

Définition 1.1.2.2 *(BCQ)*
Soit $x^* \in \Omega$. On définit l'ensemble $\Lambda(x^*) = \{(\mu,\lambda) \mid \mu \in \mathbb{R}^m, \lambda \in \mathbb{R}^p_+$ avec $\lambda_j \geq 0$ si $j \in \mathcal{A}(x^*)$ et $\lambda_j = 0$ sinon $\}$. On dit alors que la condition de qualification de base (BCQ) est satisfaite en x^* si pour $(\mu,\lambda) \in \Lambda(x^*)$ on a :

$$\sum_{i=1}^{p} \mu_i \nabla h_i(x^*) + \sum_{i \in \mathcal{A}(x^*)} \lambda_i \nabla g_i(x^*) = 0 \quad \Longleftrightarrow \quad (\mu,\lambda) = (0,0).$$

Définition 1.1.2.3 *(MFCQ)*
Etant donné le point x^* de l'ensemble admissible et l'ensemble actif $\mathcal{A}(x^*)$, on dit que la condition de qualification de Mangasarian-Fromowitz (MFCQ) est satisfaite s'il existe un vecteur $\omega \in \mathbb{R}^n$ tel que :

$$\begin{aligned} \nabla g_i(x^*)^T \omega > 0 \quad & \forall i \in \mathcal{A}(x^*), \\ \nabla h_i(x^*)^T \omega = 0 \quad & \forall i \in 1,...,p, \end{aligned}$$

et l'ensemble $\{\nabla h_i(x^*), i \in 1,...,p\}$ est linéairement indépendant.

Définition 1.1.2.4 *(LICQ)*
Etant donné le point x^* du domaine admissible et l'ensemble actif $\mathcal{A}(x^*)$, on dit que la condition de qualification d'indépendance linéaire (LICQ) est satisfaite si les gradients des contraintes actives et les gradients des contraintes d'égalité, $\{\nabla g_i(x^*), i \in$

$\mathcal{A}(x^*)\} \bigcup \{\nabla h_i(x^*), i \in 1, ..., p\}$, *sont linéairement indépendants.*

On dispose de la relation (1.2) entre les différentes conditions de qualification.

$$\text{LICQ} \Rightarrow \text{MFCQ} \Leftrightarrow \text{BCQ} \qquad (1.2)$$

Cette relation est classique. La première implication est très simple à démontrer et l'équivalence entre MFCQ et BCQ est démontrée dans (McCormick, 1967). À présent, nous possèdons tous les éléments pour introduire les conditions nécessaires d'optimalité au premier ordre en optimisation avec contraintes.

1.1.2.2 Conditions d'optimalité

Définition 1.1.2.5 *(Lagrangien)*
On associe au problème (NLP) la fonction Lagrangienne définie comme suit :

$$\begin{aligned} \mathcal{L}: \ \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^m &\to \mathbb{R} \\ (x, \mu, \lambda) &\mapsto \mathcal{L}(x, \mu, \lambda) = f(x) - \sum_{i=1}^{p} \mu_i h_i(x) - \sum_{i=1}^{m} \lambda_i g_i(x). \end{aligned}$$

Énonçons alors les conditions d'optimalité du premier ordre, plus connues sous le nom de **conditions de Karush, Kuhn et Tucker (KKT)** :

Théorème 1.1.4 *(Conditions nécessaires d'optimalité du premier ordre) Supposons que x^* est une solution locale de (NLP) et que la BCQ est satisfaite en x^*. Il existe alors un vecteur (μ^*, λ^*) (appelé multiplicateurs de Lagrange), de composantes μ_i^* pour $i \in \{1, ..., p\}$ et λ_i^* pour $i \in \{1, ..., m\}$ tel que les conditions suivantes sont satisfaites en (x^*, μ^*, λ^*) :*

$$\nabla_x \mathcal{L}(x^*, \mu^*, \lambda^*) = 0, \qquad (1.3a)$$

$$h_i(x^*) = 0 \qquad \forall i \in 1, ..., p, \qquad (1.3b)$$

$$g_i(x^*) \geq 0 \qquad \forall i \in 1, ..., m, \qquad (1.3c)$$

$$\lambda_i^* \geq 0 \qquad \forall i \in 1, ..., m, \qquad (1.3d)$$

$$\lambda_i^* g_i(x^*) = 0 \qquad \forall i \in 1, ..., m. \qquad (1.3e)$$

Ces conditions nous permettent de savoir comment la dérivée première de l'objectif et des contraintes actives sont reliées à x^*. On dispose aussi du résultat qui suit, publié dans (Gauvin, 1977).

Théorème 1.1.5 *(Conditions de Qualification et Muliplicateurs optimaux) Soit x^* un point stationnaire de (NLP). La MFCQ est satisfaite en x^* ssi l'ensemble des multiplicateurs est non-vide, fermé et borné.*

Il est intéresant de noter que d'après le théorème 1.1.5, si on suppose la LICQ au lieu de la BCQ dans le théorème 1.1.4, on sait alors que le jeu de multiplicateurs optimaux est unique. Dans le théorème 1.1.4, on sait seulement que les multiplicateurs optimaux se trouvent dans un ensemble compact.

Nous présentons maintenant les conditions d'optimalité de Fritz-John généralisées (Mangasarian & Fromowitz, 1967) qui sont un peu moins restrictives que celles de KKT (car elles ne supposent pas que des conditions de qualification soient satisfaites) :

Théorème 1.1.6 *(Conditions nécessaires d'optimalité de Fritz-John généralisées) Supposons que x^* est une soluton locale de (NLP). Il existe alors des vecteurs $\lambda^* = (\lambda_0^*, \lambda_1^*, ..., \lambda_m^*)$ et $\mu^* = (\mu_1^*, ..., \mu_p^*)$ tels que :*

$$\lambda_0^* \nabla f(x) - \sum_{i=1}^{p} \mu_i^* \nabla h_i(x^*) - \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) = 0, \tag{1.4a}$$

$$\lambda_i^* g_i(x^*) = 0 \qquad i = 1, ..., m, \tag{1.4b}$$

$$h(x^*) = 0, \tag{1.4c}$$

$$g(x^*) \geq 0, \tag{1.4d}$$

$$\lambda^* \geq 0, \tag{1.4e}$$

$$(\lambda^*, \mu^*) \neq (0, 0). \tag{1.4f}$$

Nous utiliserons ces conditions lorsque nous aborderons certains problèmes qui ne satisfont pas les conditions de qualification requises pour les conditions de KKT. Enfin, avant de poursuivre, nous allons énoncer quelques notions qui nous seront utiles.

Définition 1.1.2.6 *(Cône critique $F(\lambda^*)$)*
On définit l'ensemble $F(\lambda^*)$, appelé cône critique, de la manière suivante

$$\omega \in F(\lambda^*) \Leftrightarrow \begin{cases} \nabla h_i(x^*)^T\omega = 0, & \forall i \in 1...p, \\ \nabla g_i(x^*)^T\omega = 0, & \forall i \in 1...m \mid i \in \mathcal{A}(x^*) \text{ et } \lambda_i^* > 0, \\ \nabla g_i(x^*)^T\omega \geq 0, & \forall i \in 1...m \mid i \in \mathcal{A}(x^*) \text{ et } \lambda_i^* = 0. \end{cases} \tag{1.5}$$

Théorème 1.1.7 *(Conditions suffisantes du Second Ordre) Supposons que pour $x \in \mathbb{R}^n$ admissible il existe un vecteur λ^* de multiplicateurs de Lagrange tel que les conditions de (1.3) soient satisfaites. Supposons aussi que pour tout $\omega \neq 0$ appartenant à $F(\lambda^*)$*

$$\omega^T\nabla^2\mathcal{L}(x^*, \lambda^*)\omega > 0.$$

Alors x^* est un minimum local strict pour (NLP).

Définition 1.1.2.7 *(Complémentarité stricte)*
Soit x^* une solution locale de (NLP) et un vecteur λ^* qui vérifie les conditions de KKT. On dit que la complémentarité stricte est vérifiée en x^* si et seulement si :

$$i \in \mathcal{A}(x^*) \iff \lambda_i^* > 0 \quad \forall i \in 1, ..., m.$$

Illustration
Considérons le point $x^* = (0, 0)$ et la région admissible définie par

$$2 - (x_1 - 1)^2 - (x_2 - 1)^2 \geq 0, \tag{1.6a}$$
$$2 - (x_1 - 1)^2 - (x_2 + 1)^2 \geq 0, \tag{1.6b}$$
$$x_1 \geq 0. \tag{1.6c}$$

En x^* toutes les contraintes sont actives. En ce point, les gradients des contraintes (1.6a) et (1.6b) valent respectivement $(2, 2)$ et $(2, -2)$. De plus, le gradient de la contrainte (1.6c) vaut $(1, 0)$. Par conséquent, la LICQ n'est pas satisfaite en x^* (car les gradients des contraintes actives ne sont pas linéairement indépendants). Le vecteur

$\omega = (1, 0)$ est tel que

$$\begin{aligned}(2, 2)^T(1, 0) = 5 &> 0\\ (2, -2)^T(1, 0) = 2 &> 0.\end{aligned}$$

La MFCQ est donc satisfaite en x^*. Prenons maintenant la fonction objectif suivante

$$(x_1 + 2)^2 + x_2^2$$

et essayons de la minimiser sur le domaine admissible précédent. La solution évidente est $(x_1, x_2) = (0, 0)$. Puisque la MFCQ est satisfaite en ce point et qu'il est optimal, il vérifie les conditions de KKT. Cela s'écrit :

$$\begin{aligned}\begin{bmatrix} 4 \\ 0 \end{bmatrix} - \lambda_1 \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \lambda_2 \begin{bmatrix} 2 \\ -2 \end{bmatrix} - \lambda_3 \begin{bmatrix} 1 \\ 0 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix},\\ 2 - 2 &\geq 0,\\ 2 - 2 &\geq 0,\\ 0 &\geq 0,\\ (2 - 2)\lambda_1 &= 0,\\ (2 - 2)\lambda_2 &= 0,\\ 0\lambda_3 &= 0,\\ \lambda_1 &\geq 0,\\ \lambda_2 &\geq 0,\\ \lambda_3 &\geq 0.\end{aligned}$$

On en tire

$$\begin{aligned}4 - 2\lambda_1 - 2\lambda_2 - \lambda_3 &= 0,\\ 0 - 2\lambda_1 + 2\lambda_2 &= 0,\end{aligned}$$

et donc

$$\begin{aligned}4(1 - \lambda_1) &= \lambda_3,\\ \lambda_1 &= \lambda_2.\end{aligned}$$

L'ensemble des multiplicateurs est donc constitué du segment

$$L = \{(\lambda, \lambda, 4(1-\lambda)) : \lambda \in [0,1]\} \subset \mathbb{R}^3,$$

qui est bien compact, comme on peut le voir sur la figure 1.1 sur laquelle l'ensemble L est représenté en gras.

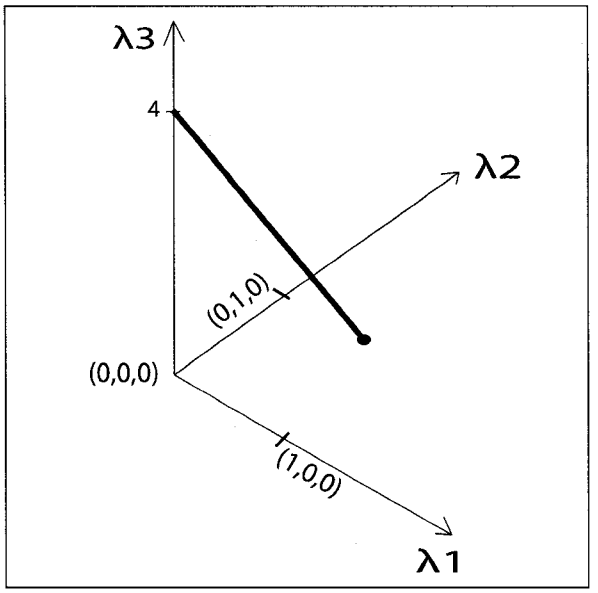


Figure 1.1 – Représentation de l'ensemble des multiplicateurs optimaux

1.1.3 Problèmes dégénérés

Dans certains problèmes, il peut arriver que les conditions de qualification ne soient pas satisfaites. Dans le cas où la MFCQ est violée en un point solution x^*, cette solution est dite dégénérée. Si un problème possède au moins une solution dégénérée, on écrira dans la suite que le problème est dégénéré. Regardons sans plus tarder quelques exemples.

Exemple 1 *: MFCQ non-satisfaite*

Soit le problème HS013, issu de la collection de (Hock & Schittkowski, 1981) :

$$\begin{aligned} \underset{(x_1,x_2)}{\text{minimiser}} \quad & (x_1-2)^2+x_2^2 \\ \text{s.c. :} \quad & (1-x_1)^3 \geq x_2, \\ & x_1 \geq 0, \\ & x_2 \geq 0. \end{aligned}$$

Le point optimal pour ce problème est $x^* = (1, 0)$. Au point optimal, le gradient de la première contrainte est (0, -1) et celui de la deuxième contrainte est (0,1). On ne peut pas trouver ω tel que $(0,1)^T\omega > 0$ et $(0,-1)^T\omega > 0$ donc la MFCQ n'est pas satisfaite en x^* (sans avoir besoin de considérer la troisième contrainte). On peut représenter cela schématiquement par la figure 1.2.

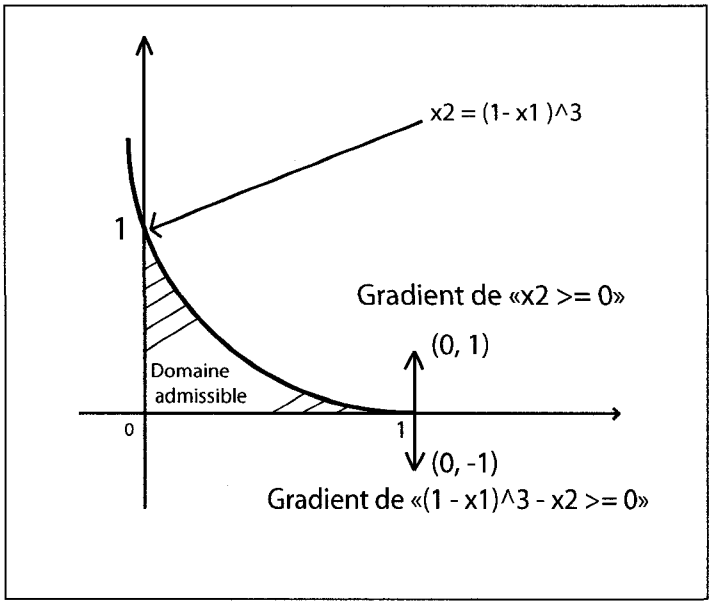


Figure 1.2 – HS013 et la MFCQ

Exemple 2 : *Programmes Mathématiques avec Contraintes d'Équilibre (MPEC)*
Les MPEC [1] se présentent généralement sous la forme suivante :

[1]MPEC : Mathematical Programs with Equilibrium Constraints

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimiser}} & f(x) \\ \text{s.c. :} & F^T x \geq 0, \\ & G^T x \geq 0, \\ & (F^T x) \perp (G^T x), \end{array} \tag{MPEC}$$

où $f : \mathbb{R}^n \to \mathbb{R}$ est de classe $\mathcal{C}^2$, F et G sont des matrices $n \times p$. La notation $(F^T x) \perp (G^T x)$ signifie qu'on multiplie terme à terme les vecteurs $(F^T x)$ et $(G^T x)$ et le vecteur résultant de cette opération est nul, *ie* $(F^T x)_i (G^T x)_i = 0$ pour tout $i \in 1, ..., p$. Remarquons que d'autres contraintes peuvent éventuellement s'ajouter au modèle (MPEC). Dans la suite, nous proposons une série de propriétés, définitions et théorèmes permettant au lecteur de mieux cerner les particularités de ces programmes mathématiques.

Propriété. Un problème de type MPEC est dégénéré puisqu'il ne satisfait la MFCQ (à fortiori la LICQ) en aucun point de son domaine admissible.

Preuve : Supposons que le domaine admissible pour (MPEC) ne soit pas vide. Soit x un point admissible. On note $F_i(x) = (F^T x)_i$ (respectivement $G_i(x) = (G^T x)_i$) c'est-à-dire la i-ème composante du vecteur $F^T x$ (respectivement $G^T x$).
On sait que $F_i(x)G_i(x) = 0, \forall i \in 1, ..., p$. On distingue alors trois cas : $F_i(x) = 0$ et $G_i(x) > 0$, $G_i(x) = 0$ et $F_i(x) > 0$ ou $F_i(x) = G_i(x) = 0$. Supposons maintenant que la MFCQ soit satisfaite en x.

Premier cas : $F_i(x) = 0$ et $G_i(x) > 0$

Puisque la MFCQ est satisfaite en x et que $F_i(x) = 0$, il existe une direction ω telle que

$$\nabla F_i(x)^T \omega > 0, \tag{1.7}$$

$$\nabla (F_i G_i)(x)^T \omega = 0.$$

Or $\nabla (F_i G_i)(x)^T \omega = F_i(x) \nabla G_i(x)^T \omega + G_i(x) \nabla F_i(x)^T \omega$ donc

$$F_i(x) \nabla G_i(x)^T \omega + G_i(x) \nabla F_i(x)^T \omega = 0.$$

De plus, puisque $F_i(x) = 0$ on a

$$G_i(x) \nabla F_i(x)^T \omega = 0.$$

Or $G_i(x) > 0$ ce qui contredit (1.7).

Deuxième cas : $G_i(x) = 0$ et $F_i(x) > 0$

Même raisonnement que dans le premier cas en inversant le rôle de F et G.

Troisième cas : $F_i(x) = G_i(x) = 0$

On a alors $\nabla(F_iG_i)(x)^T = F_i(x)\nabla G_i(x)^T + G_i(x)\nabla F_i(x)^T = 0$ ce qui contredit la condition d'indépendance linéaire des gradients des contraintes d'égalité. □

Un domaine admissible typique pour un MPEC à deux variables ressemble au schéma de la figure 1.3.

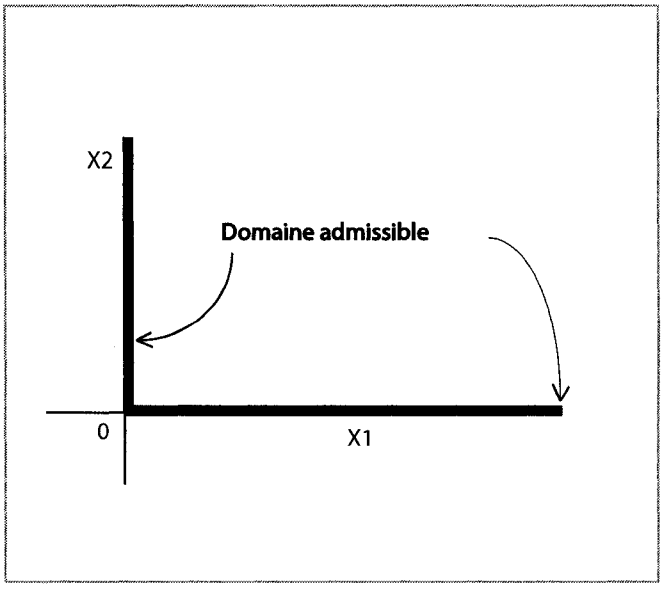


Figure 1.3 – Domaine admissible typique d'un MPEC

Illustration

Considérons le problème *scale1* issu de la collection MacMPEC (Leyffer, 2005) :

$$\begin{array}{ll} \underset{(x_1,x_2)}{\text{minimiser}} & (100x_1 - 1)^2 + (x_2 - 1)^2 \\ \text{s.c. :} & x_1 \geq 0, \\ & x_2 \geq 0, \\ & x_1 x_2 = 0. \end{array}$$

On constate que :

- son domaine admissible est exactement celui décrit par la figure 1.3,
- la MFCQ n'est satisfaite en aucun point admissible.
  En effet, les gradients des contraintes sont $(1, 0)$ pour la première, $(0, 1)$ pour la seconde et (x_2, x_1) pour la troisième. Il n'existe donc pas de $\omega = (\omega_1, \omega_2)$ tel que $(1, 0)^T\omega > 0$ et $(0, 1)^T\omega > 0$ et $(x_2, x_1)^T\omega = 0$ (car pour avoir les deux premières conditions il faut $\omega_1 > 0$ et $\omega_2 > 0$ et alors la troisième ne peut pas tenir sauf en $(0, 0)$ qui ne convient pas pour cause de violation de l'indépendance linéaire des gradients des contraintes d'égalité).

On vient de remarquer que les MPEC ne satisfont la MFCQ en aucun point de leur domaine admissible. Pourtant ils admettent des multiplicateurs tels que certaines conditions d'optimalité puissent être vérifiées.

Définition 1.1.3.1 *(Lagrangien pour un MPEC)*
Le Lagrangien pour (MPEC) se définit par

$$\mathcal{L}(x, \mu, \tau, \nu) = f(x) - \tau^T G^T x - \nu^T F^T x, \tag{1.8}$$

Le résultat suivant est établi dans (Scheel & Scholtes, 2000).

Théorème 1.1.8 *(Stationarité forte) Un point admissible x pour (MPEC) est dit fortement stationnaire si et seulement s'il existe des multiplicateurs (τ, ν) tels que :*

$$\begin{array}{l} \nabla\mathcal{L}(x, \tau, \nu) = 0, \\ \tau \perp G^T x \geq 0, \\ \nu \perp F^T x \geq 0, \\ \qquad \tau_i \geq 0 \quad i \in I_G \bigcap I_F, \\ \qquad \nu_i \geq 0 \quad i \in I_G \bigcap I_F, \end{array} \tag{1.9}$$

où I_G (respectivement I_F) est l'ensemble des indices pour lesquels $G_i^T x = 0$ (respectivement $F_i^T x = 0$).

Remarque : On peut souligner la différence entre les conditions du théorème 1.1.8 et les conditions classiques de KKT 1.1.4. La différence réside dans le fait que pour $i \notin I_G \bigcap I_F$, les multiplicateurs τ_i et ν_i peuvent prendre n'importe quel signe.

Le théorème précédent est très intéressant lorsqu'on cherche à aborder les MPEC d'un point de vue numérique. En effet, on sait que pour un point optimal, les conditions de KKT classiques ne sont pas obligatoirement satisfaites (car la MFCQ n'est pas satisfaite) mais on peut alors les remplacer par les conditions (1.9). Notons aussi que l'existence de multiplicateurs pour un point fortement stationnaire est très important car il justifie l'emploi de méthodes numériques fondées sur l'utilisation de ces multiplicateurs (nous verrons que c'est le cas de l'algorithme de la section 1.2.5).

Exemple 3 *: Problèmes Mathématiques avec Contraintes Évanescentes (MPVC [2])*

Les MPVC peuvent s'écrire sous la forme

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimiser}} & f(x) & \\ \text{s.c. :} & h_i(x) = 0 & \forall i \in I_h, \\ & g_i(x) \leq 0 & \forall i \in I_g, \\ & H_i(x) \geq 0 & \forall i = 1...l, \\ & H_i(x)G_i(x) \leq 0 & \forall i = 1...l, \end{array} \qquad \text{(MPVC)}$$

où $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^{|I_h|}$, $g : \mathbb{R}^n \to \mathbb{R}^{|I_g|}$, $H_i : \mathbb{R}^n \to \mathbb{R}$ et $G_i : \mathbb{R}^n \to \mathbb{R}$ sont des fonctions de classe $\mathcal{C}^2$. Notons qu'on peut alors comprendre l'appellation «contrainte évanescente». Cette contrainte correspond à la dernière du modèle précédent. En effet, on remarque que si $H_i(x) = 0$ cette contrainte est automatiquement satisfaite puisqu'alors $H_i(x)G_i(x) = 0$. De plus alors, $G_i(x)$ n'a plus aucune contrainte de signe.

Illustration

On peut illustrer le domaine admissible typique pour un MPVC par le dessin de la

[2]MPVC : Mathematical Programs with Vanishing Constraints

figure 1.4 qui correspond aux contraintes

$$x_1 \geq 0,$$
$$x_1 x_2 \leq 0.$$

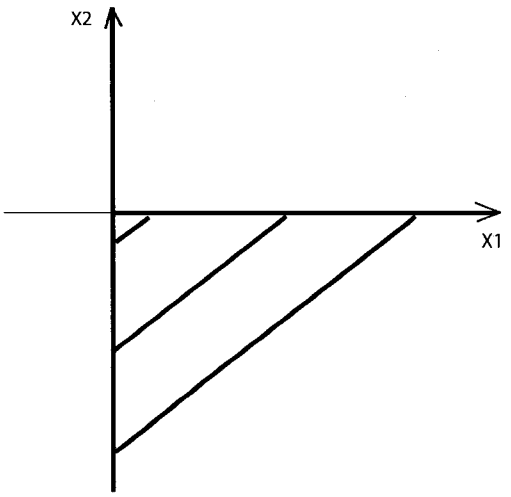


Figure 1.4 – Domaine admissible typique d'un MPVC

Nous verrons que les problèmes auxquels nous nous sommes intéressés peuvent parfois s'écrire sous forme de MPVC.

1.2 Méthodes de pénalisation ℓ_1

Afin de résoudre les problèmes présentés précédemment, nous allons utiliser une méthode développée dans (Gould, Orban, & Toint, 2003). Cette méthode se fonde à la fois sur les principes des méthodes de région de confiance, de points intérieurs, de pénalisation $\ell 1$ ainsi que des variables élastiques. C'est la raison pour laquelle, dans un premier temps, nous présentons les résultats fondamentaux concernant ces différentes stratégies.

1.2.1 Méthodes de région de confiance

On considère le problème sans contrainte

$$\underset{x\in\mathbb{R}^n}{\text{minimiser}}\ f(x), \tag{1.10}$$

où $f : \mathbb{R}^n \to \mathbb{R}$ est de classe $\mathcal{C}^2$. De plus on suppose que f est fortement non-linéaire et/ou potentiellement coûteuse à évaluer. La philosophie des méthodes de région de confiance est de résoudre itérativement le problème (1.10) en considérant, à partir d'un itéré, toutes les directions possibles mais en limitant la longueur du pas vers le prochain itéré.

À l'itération k, l'idée d'une méthode de région de confiance est de remplacer f par un modèle, typiquement quadratique, m_k facile à évaluer. Cependant ce modèle n'est valable que localement autour de l'itéré actuel x_k. On définit alors une région $\mathcal{B}_k$ autour de x_k à l'intérieur de laquelle on pense que le modèle a du sens : c'est la région de confiance. Mathématiquement, on définit la région de confiance comme la boule fermée

$$\mathcal{B}_k \equiv \mathcal{B}(x_k, \delta_k) = \{x_k + s \in \mathbb{R}^n \mid ||s|| \leq \delta_k\},$$

où $\delta_k > 0$ est appelé le rayon de la région de confiance et où $||.||$ représente une norme donnée sur $\mathbb{R}^n$.

Au lieu de minimiser f à partir de x_k, on minimise le modèle m_k dans la boule $\mathcal{B}_k$. On trouve ainsi un pas s. Pour être pris en compte, ce pas doit *suffisamment* faire décroître la valeur du modèle. Si ce pas produit également une décroissance de la fonction, on accepte le pas et on passe à l'itération suivante en augmentant peut-être δ_k. Sinon, le pas est rejeté et on diminue le rayon. Dans ce cas, cela signifie qu'on a fait confiance au modèle dans une trop large mesure et c'est pour cela qu'on restreint la taille de la région de confiance à l'itération suivante.

En pratique, on peut utiliser le modèle suivant (c'est le choix que nous avons adopté dans nos travaux) :

$$m_k(x_k + s) = f(x_k) + \nabla f(x_k)^T s + \frac{1}{2} s^T H_k s,$$

où $H_k = \nabla^2 f(x_k)$, ou bien une approximation symétrique de cette matrice. Notons qu'une des forces des méthodes de région de confiance est de ne pas nécessiter que la

matrice H_k soit définie positive.

La condition de décoissance suffisante est précisée de la façon suivante : on calcule la décroissance obtenue en deux points de référence de la région de confiance. Le premier de ces points est le point de Cauchy, x_k^C, minimum du modèle suivant la direction de plus forte pente. Le second est appelé le point propre, x_k^P et correspond à un minimum du modèle selon une direction de courbure suffisamment négative s'il en existe une. On dit qu'on a une décroissance suffisante si la condition suivante est satisfaite

$$m_k(x_k) - m_k(x_k + s) \geq \theta(m_k(x_k) - \min\{m_k(x_k^C), m_k(x_k^P)\}), \tag{1.11}$$

où $0 < \theta < 1$ est fixé et indépendant de l'itération à laquelle on se trouve.

Finalement, on peut rassembler toutes ces informations et présenter l'algorithme 1.2.1, typique d'une méthode de région de confiance.

Remarque : En ce qui concerne la norme utilisée, on peut utiliser la norme euclidienne mais on peut aussi utiliser une norme de mise à l'échelle associée à une matrice M, définie positive (voir section 2.2.2 du chapitre 2). Les normes considérées doivent être uniformément équivalentes. Notons que dans la pratique, à l'étape 1 de l'algorithme 1.2.1, on résout le problème d'optimisation suivant

$$\begin{array}{ll} \underset{s\in\mathbb{R}^n}{\text{minimiser}} & \nabla f(x)^T s + \frac{1}{2} s^T H_k s \\ \text{s.c. :} & ||s|| \leq \delta_k, \end{array} \tag{1.13}$$

où on utilise les mêmes notations que précédemment. Concernant les méthodes de région de confiance en général, voici ce qu'il faut retenir :

On résout itérativement le problème (1.13) jusqu'à satisfaction d'un critère d'arrêt. On obtient une suite d'itéré $\{x_k\}_{k\in\mathbb{N}}$. Soient les hypothèses suivantes :

(a) la fonction f est bornée inférieurement,

(b) la matrice Hessienne de f est uniformément bornée sur un ensemble qui contient la suite des itérés,

(c) le modèle coïncide avec f jusqu'au premier ordre *i.e.*, $m_k(x_k) = f(x_k)$ ainsi que

Algorithme 1.2.1 Méthode de région de confiance

Etape 0. Initialisation - On donne $x_0 \in \mathbb{R}^n$ et un rayon de région de confiance $\delta_0 > 0$ ainsi que des paramètres η_1, η_2, α_1 et α_2 qui satisfont

$$0 \leq \eta_1 < \eta_2 < 1 \quad \text{et} \quad 0 < \alpha_1 < 1 < \alpha_2.$$

Calculer $f(x_0)$ et poser $k = 0$.

Etape 1. Calcul du pas - Définir un modèle $m_k(x_k + s)$ de $f(x_k + s)$ dans $\mathcal{B}_k$ et calculer un pas $s_k \in \mathcal{B}_k$ qui satisfait (1.11)

Etape 2. Acceptation du candidat - Calculer $f(x_k + s_k)$ et

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}.$$

Si $\rho_k \geq \eta_1$ alors poser $x_{k+1} = x_k + s_k$;
Sinon poser $x_{k+1} = x_k$.

Etape 3. Mise a jour du rayon - Poser

$$\delta_{k+1} = \begin{cases} \alpha_1 ||s_k|| & si \quad \rho_k < \eta_1, \\ \alpha_1 \delta_k & si \quad \eta_1 \leq \rho_k < \eta_2, \\ \max[\alpha_2 ||s_k||, \delta_k] & si \quad \eta_2 \leq \rho_k. \end{cases} \tag{1.12}$$

Incrémenter k de un et aller à Etape 1.

$$\nabla m_k(x_k) = \nabla f(x_k),$$

(d) le paramètre η_1 est strictement positif.

On dispose alors du theorème suivant (voir (Conn, Gould, & Toint, 2000)) :

$$\lim_{k \to \infty} ||\nabla f(x_k)|| = 0.$$

Cela signifie que tous les points d'adhérence sont des points stationnaires pour le problème (1.10).

1.2.2 Méthodes de points intérieurs

On s'intéresse à un problème ayant la forme suivante :

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimiser}} \quad & f(x) \\ \text{s.c. :} \quad & g(x) \geq 0, \end{aligned} \tag{1.14}$$

où $f : \mathbb{R}^n \to \mathbb{R}$ et $g : \mathbb{R}^n \to \mathbb{R}^m$ sont de classe $\mathcal{C}^2$. On suppose que l'intérieur strict du domaine admissible est non vide. On introduit alors une fonction barrière, b, continue sur l'intérieur strict du domaine admissible et telle que :

$$\lim_{g(x) \to 0} b(x) = +\infty.$$

Un exemple de fonction barrière est la barrière logarithmique définie par

$$b(x) = -\sum_{i=1}^{m} \ln(g_i(x)).$$

Une méthode de points intérieurs consiste alors à s'intéresser à une suite de sous-problèmes sans contraintes, $\{P_\mu\}_{\mu > 0}$ définis par :

$$\underset{x \in \mathbb{R}^n}{\text{minimiser}} \ \phi^B(x, \mu) = f(x) + \mu b(x), \tag{1.15}$$

qu'on résout itérativement en faisant décroître μ vers zéro. Ainsi, sous certaines conditions, on a une suite $\{x(\mu)\}_{\mu > 0}$ de solutions des problèmes $\{P_\mu\}_{\mu > 0}$. Pour tout $\mu > 0$, on a par définition $g_i(x(\mu)) > 0$. Nous expliquons dans ce qui suit pourquoi on agit

ainsi.

Les conditions d'optimalité classiques de KKT pour (1.14) sont

$$\begin{aligned} \nabla f(x) - J^T(x)\lambda = 0, & \\ g_i(x) \geq 0 \quad \forall i \in 1...m, & \\ \lambda_i^* \geq 0 \quad \forall i \in 1...m, & \\ \lambda_i^* g_i(x) = 0 \quad \forall i \in 1...m. & \end{aligned} \tag{1.16}$$

où $J(x)$ est la Jacobienne des contraintes évaluées en x. Regardons les propriétés d'une solution $x(\mu)$ de (1.15). Un minimum de ϕ^B vérifie

$$\nabla f(x(\mu)) - \sum_i \frac{\mu}{g_i(x(\mu))} \nabla g_i(x(\mu)) = 0. \tag{1.17}$$

Posons $\lambda_i(\mu) = \frac{\mu}{g_i(x(\mu))}$. Puisque $g_i(x(\mu)) > 0$ on a aussi $\lambda_i(\mu) > 0$. L'équation (1.17) se réecrit de manière équivalente

$$\begin{aligned} \nabla f(x(\mu)) - J^T(x(\mu))\lambda(\mu) = 0, & \\ \lambda_i(\mu) g_i(x(\mu)) = \mu \quad \forall i \in 1...m. & \end{aligned} \tag{1.18}$$

Par continuité, on peut s'attendre à ce que lorsque $\mu \downarrow 0$, un point qui vérifie (1.18) tend à vérifier les conditions (1.16) qui sont les conditions classiques de KKT. On dispose alors de (Fiacco & McCormick, 1968, théorème 12) :

Théorème 1.2.1 *(Existence et unicité de la trajectoire centrale) Supposons que le domaine admissible de (1.14) ait un intérieur strict non-vide. Soit x^* un minimum local de (1.14) tel que :*

- *x^* vérifie les conditions de KKT du premier ordre pour un jeu de multiplicateurs λ^*.*
- *Les conditions suffisantes du second ordre, la complémentarité stricte et la LICQ sont satisfaites en (x^*, λ^*).*

Alors les assertions suivantes sont vraies :

***(i)** Il existe une unique fonction $x(\mu)$ définie pour tout $\mu > 0$ suffisamment petit, de classe $\mathcal{C}^1$ et telle que $x(\mu)$ est un minimum local strict de $\phi^B(x, \mu)$ et $\lim_{\mu \to 0} x(\mu) = x^*$.*

(ii) *Pour la fonction $x(\mu)$ de (i), les estimateurs des multiplicateurs de Lagrange, $\lambda_i(\mu) := \frac{\mu}{c_i(x(\mu))}$ convergent vers λ_i^* quand $\mu \downarrow 0$.*
(iii) *Le Hessien $\nabla^2_{xx}\phi^B(x,\mu)$ est défini positif pour tout μ suffisament petit.*

Le nom du théorème provient du fait que la trajectoire C^P définie par

$$C^P \equiv \{x(\mu) \mid \mu > 0\},$$

est appelée *trajectoire centrale primale.* La *trajectoire centrale primale-duale* est

$$C^{PD} \equiv \{(x(\mu), \lambda(\mu)) \mid \mu > 0\}.$$

Exemple
Soit le problème suivant :

$$\begin{array}{ll} \text{minimiser} & x_1^2 + x_2^2 \\ \text{s.c :} & x_1 \geq 2. \end{array} \tag{1.19}$$

Résolvons ce problème de deux manières différentes.

a) Conditions de KKT : Les conditions de KKT pour le problème (1.19) s'écrivent

$$\begin{gathered} \nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \\ x_1^* \geq 2, \\ \lambda^* \geq 0, \\ \lambda^*(x_1^* - 2) = 0, \end{gathered}$$

où $\nabla_x \mathcal{L}(x^*, \lambda^*) = 0$ est équivalent à :

$$\begin{pmatrix} 2x_1^* - \lambda^* \\ 2x_2^* \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

La dernière condition du système de KKT peut se réécrire $2x_1^*(x_1^* - 2) = 0$ car

$\lambda^* = 2x_1^*$. Donc à l'optimalité on a :

$$\begin{pmatrix} x_1^* \\ x_2^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}.$$

b) Trajectoire centrale primale : Cherchons à présent la trajectoire centrale primale pour le problème (1.19). On forme d'abord le problème barrière :

$$(P_\mu) \text{ minimiser } \quad \phi_\mu^B(x) = x_1^2 + x_2^2 - \mu \ln(x_1 - 2).$$

On résout alors $\nabla_x \phi_\mu^B = 0$. On obtient :

$$\begin{pmatrix} x_1^\mu \\ x_2^\mu \end{pmatrix} = \begin{pmatrix} \frac{2+\sqrt{4+2\mu}}{2} \\ 0 \end{pmatrix}. \tag{1.20}$$

D'après (1.20) il est clair que $\lim_{\mu \to 0} x_1^\mu = 2$ donc on a :

$$\lim_{\mu \to 0} \begin{pmatrix} x_1^\mu \\ x_2^\mu \end{pmatrix} = \begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}.$$

Quand au λ, on a

$$\lambda(\mu) = \frac{\mu}{x_1^\mu - 2} = \frac{\mu}{\frac{1}{2}\sqrt{4+2\mu} - 1} = \sqrt{4+2\mu} + 2.$$

Quand $\mu \to 0$, on obtient le même résultat qu'auparavant ($\lambda^* = 4$).

À présent, il faut faire la distinction entre deux types de méthodes de points intérieurs : les méthodes *primales* et les méthodes *primales-duales*. La distinction a lieu au niveau de la manière dont on considère les multiplicateurs. Dans le cas *primal*, on cherche les solutions, en termes de $x(\mu)$, au système (1.15). On en déduit la valeur des multiplicateurs via la relation $\lambda_i(\mu) = \frac{\mu}{g_i(x(\mu))}$. En revanche, dans le cas *primal-dual*, on résout (1.18) mais on considére les multiplicateurs $\lambda(\mu)$ comme des variables à part entière. D'un point de vue numérique, il s'est avéré que les méthodes *primales-duales* sont nettement supérieures aux méthodes primales comme on peut le constater dans

(Conn, Gould, Orban, & Toint, 2000), (Byrd, Gilbert, & Nocedal, 2000a), (Byrd, Hribar, & Nocedal, 1999a), (Forsgren, Gill, & Wright, 2002), (Vanderbei & Shanno, 1999) ou encore (Wright, 1992), ainsi que dans les références citées dans ces articles.

Notation 1.2.2.1 *Soit $a \in \mathbb{R}^p$. On note A la matrice diagonale de $\mathbb{R}^{p \times p}$, dont les termes diagonaux sont $A_{ii} = a_i, \forall i \in 1...p$.*

Notation 1.2.2.2 *On note e le vecteur de 1 : $e = (1, 1, ..., 1)$.*

On peut à présent synthétiser le fonctionnement des méthodes de points intérieurs par deux algorithmes types, 1.2.2 et 1.2.3.

Algorithme 1.2.2 Méthode de points intérieurs primale

Etape 0. Choisir x_0 admissible, $\mu_0 > 0$ et un paramètre $0 < \alpha < 1$;
Poser $k = 0$.

Etape 1. Si des conditions d'optimalité pour (1.14) sont satisfaites :
STOP.
Sinon aller à l'étape 2.

Etape 2. Trouver un point $x_k = x(\mu_k)$ qui résout (1.15) de manière approchée
Poser $\lambda_k = \mu_k G^{-1}(x_k)e$;
Incrémenter k de un ;
Poser $\mu_k = \alpha \mu_{k-1}$;
Aller à l'étape 1.

Algorithme 1.2.3 Méthode de points intérieurs primale-duale

Etape 0. Choisir x_0 admissible, $\mu_0 > 0$ et un paramètre $0 < \alpha < 1$;
Poser $k = 0$.

Etape 1. Si des conditions d'optimalité pour (1.14) sont satisfaites :
STOP.
Sinon aller à l'étape 2.

Etape 2. Trouver (x_k, λ_k) qui résout (1.18) de manière approchée ;
Incrémenter k de un ;
Poser $\mu_k = \alpha \mu_{k-1}$;
Aller à l'étape 1.

Essayons de comprendre en quoi les deux méthodes diffèrent. La méthode primale essaie de résoudre (1.15) de façon approchée par la méthode de Newton alors que la

méthode primale-duale applique la méthode de Newton pour les systèmes d'équations (1.18).

1.2.3 Fonction de mérite $\ell 1$

On s'intéresse dans ce cas au problème (NLP) mais pour simplifier la notation, on note c_i pour $i \in \mathcal{E}$ les contraintes d'égalité et c_i pour $i \in \mathcal{I}$ les contraintes d'inégalité :

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimiser}} & f(x) \\ \text{s.c. :} & c_{\mathcal{E}}(x) = 0, \\ & c_{\mathcal{I}}(x) \geq 0, \end{array} \tag{1.21}$$

où $f : \mathbb{R}^n \to \mathbb{R}$ et $c_i : \mathbb{R}^n \to \mathbb{R}$, $i \in \mathcal{E} \cup \mathcal{I}$ sont de classe $\mathcal{C}^2$.

Dans (1.21), on est en présence de deux objectifs conflictuels : minimiser f et minimiser la violation des contraintes. En un point x, on peut écrire la violation des contraintes de la façon suivante

$$V(x) = \sum_{i\in\mathcal{E}} |c_i(x)| + \sum_{i\in\mathcal{I}} \max(-c_i(x), 0).$$

Ce qu'on veut, c'est trouver un moyen pour minimiser f et V en même temps. Une méthode pour résoudre ce genre de problème est d'avoir recours à une fonction auxiliaire appelée **fonction de mérite**. Cette fonction combine les deux objectifs et se présente, par exemple, sous la forme :

$$\Phi(x, \nu) \equiv f(x) + \nu V(x),$$

où ν est un paramètre strictement positif. Pour la fonction V définie ci-dessus, la fonction de mérite est appelée $\ell 1$. Il existe d'autres manières d'exprimer la violation des contraintes et la fonction de mérite porte alors un nom différent. La fonction de mérite $\ell 1$ est une fonction de mérite *exacte*. La définition suivante définit ce terme.

Définition 1.2.3.1 *(Fonction de mérite exacte)*
La fonction de mérite $\Phi(x, \nu)$ est dite exacte s'il existe un scalaire ν^* tel que pour tout $\nu \in [\nu^*, +\infty[$, toute solution de (1.21) est un minimum local de $\Phi(x, \nu)$.

Le principe d'un algorithme de pénalisation est de considérer la suite de problèmes $\{Q_\nu\}_{\nu>0}$ sans contrainte suivants :

$$\underset{x\in\mathbb{R}^n}{\text{minimiser}}\ \Phi(x,\nu) = f(x) + \nu\sum_{i\in\mathcal{E}}|c_i(x)| + \nu\sum_{i\in\mathcal{I}}\max(-c_i(x),0) \tag{1.22}$$

et de les résoudre itérativement en faisant croître ν si besoin. Remarquons qu'à l'intérieur du domaine réalisable, $\Phi(x,\nu) = f(x)$ alors qu'à l'extérieur du domaine $\Phi(x,\nu) > f(x)$.

Exemple

Soit le problème :

$$\begin{aligned} \underset{x\in\mathbb{R}}{\text{minimiser}} \quad & f(x) = x^2 \\ \text{s.c. :} \quad & x \leq 2, \\ & x \geq -1. \end{aligned} \tag{1.23}$$

On a alors

$$\Phi(x,\nu) = x^2 + \nu(\max(x-2,0) + \max(-1-x,0)).$$

Le domaine réalisable est $\Omega = [-1,2]$. Sur cet intervalle, les fonctions f et Φ sont donc confondues. En revanche, à l'extérieur du domaine admissible, on a

sur $]2,+\infty[$: $\Phi(x,\nu) = x^2 + \nu(x-2)$,

sur $]-\infty,-1[$: $\Phi(x,\nu) = x^2 + \nu(-1-x)$.

On remarque qu'en $x=-1$ et $x=2$, la fonction de mérite n'est pas différentiable. On peut observer graphiquement ce phénomène quand ν est grand. Par exemple, dans le cas où $\nu = 1000$ on obtient la figure 1.5.

On peut maintenant donner un algorithme de pénalisation type, l'algorithme 1.2.4. Dans la section suivante, on s'intéresse de plus près au manque de différentiabilité de (1.22).

1.2.4 Méthode élastique

Avant de rentrer dans la théorie de ces méthodes, essayons d'en comprendre intuitivement le principe. L'idée de ce type de méthode est assez simple. On introduit des variables, appelées variables élastiques, qui permettent de «relâcher» un peu les

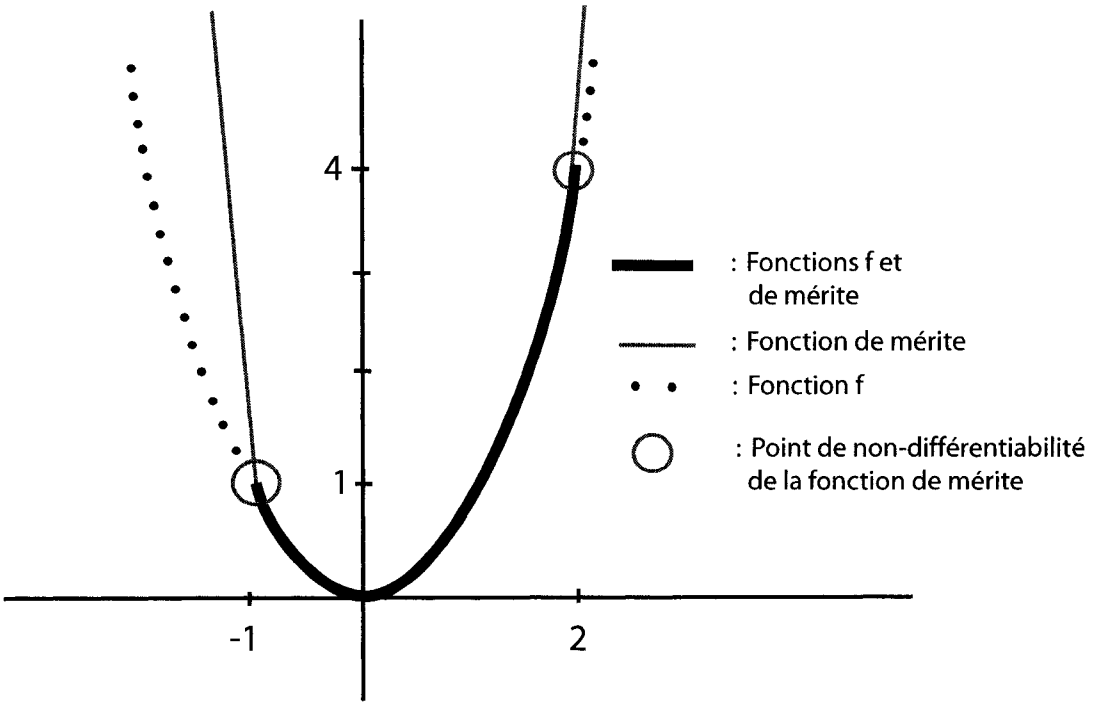


Figure 1.5 – Fonction f et Φ dans le cas du problème (1.23)

Algorithme 1.2.4 Pénalisation $\ell 1$

Etape 0. Choisir x^0 et $\nu^0 > 0$; poser $k = 0$.

Etape 1. Si des conditions d'optimalité (théorème 1.1.4 par exemple) pour (1.21) sont satisfaites : STOP.
Sinon aller à l'étape 2.

Etape 2. Trouver un point $x^k \approx x(\nu^k)$ qui résout (1.22) de manière approchée.
Incrémenter k.
Poser $\nu^k = \beta\nu^{k-1}$ avec $1 < \beta$

contraintes c'est-à-dire d'agrandir le domaine admissible. Le problème modifié est donc moins contraignant et de ce fait, il a des chances d'être plus simple à résoudre. Si on appelle s les variables élastiques, on peut représenter la situation par la figure 1.6. Le but est alors de trouver un mécanisme pour faire en sorte que ces variables diminuent au cours des itérations et soient nulles à la fin pour que le domaine admissible final corresponde à celui du problème initial. L'algorithme que nous avons utilisé, appelé PSUPERB, est une méthode élastique avec pénalisation $\ell 1$. Voyons comment il fonctionne.

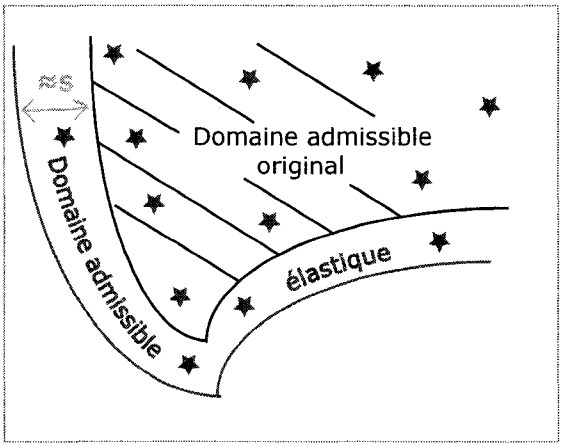


Figure 1.6 – Transformation élastique

On considère le problème suivant :

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimiser}} & f(x) \\ \text{s.c. :} & c_{\mathcal{E}}(x) = 0, \\ & c_{\mathcal{I}}(x) \geq 0, \end{array} \tag{1.24}$$

où $f : \mathbb{R}^n \to \mathbb{R}$, $c_{\mathcal{E}} : \mathbb{R}^n \to \mathbb{R}^{n_{\mathcal{E}}}$, $c_{\mathcal{I}} : \mathbb{R}^n \to \mathbb{R}^{n_{\mathcal{I}}}$ sont des fonctions de classe $\mathcal{C}^2$. On note $n_{\mathcal{C}} = n_{\mathcal{E}} + n_{\mathcal{I}}$. On utilise alors une pénalisation $\ell 1$ et on transforme donc le problème de la manière suivante :

$$\underset{x\in\mathbb{R}^n}{\text{minimiser}} \quad \Phi(x,\nu) = f(x) + \nu \sum_{i\in\mathcal{E}} |c_i(x)| + \nu \sum_{i\in\mathcal{I}} \max(-c_i(x), 0), \tag{1.25}$$

où $\nu > 0$ est le paramètre de pénalité. Ce problème n'est pas différentiable sur la frontière de l'ensemble admissible à cause des fonctions valeur absolue et max. Pour remédier à ce problème, on introduit alors les *variables élastiques*. En effet, on remarque que tout nombre réel, x, peut s'écrire de manière unique :

$$x = r_i - s_i \text{ et } |x| = r_i + s_i,$$

où $r_i \geq 0$ représente la partie positive de x et $s_i \geq 0$ sa partie négative. On peut alors écrire

$$\begin{cases} |c_i(x)| = r_i + s_i & \text{avec} \quad c_i(x) = r_i - s_i \quad \text{et} \quad (r_i, s_i) \in \mathbb{R}^2_+ \quad \forall i \in \mathcal{E}, \\ \max(-c_i(x), 0) = s_i & \text{avec} \quad c_i(x) = r_i - s_i \quad \text{et} \quad (r_i, s_i) \in \mathbb{R}^2_+ \quad \forall i \in \mathcal{I}, \end{cases}$$

qu'on peut reformuler, en éliminant r_i :

$$\begin{cases} |c_i(x)| = c_i(x) + 2s_i & \text{avec} \quad c_i(x) + s_i \geq 0 \quad \text{et} \quad s_i \geq 0 \quad \forall i \in \mathcal{E}, \\ \max(-c_i(x), 0) = s_i & \text{avec} \quad c_i(x) + s_i \geq 0 \quad \text{et} \quad s_i \geq 0 \quad \forall i \in \mathcal{I}. \end{cases}$$

Les variables s sont les *variables élastiques*. La reformulation du problème (1.25) est alors :

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n, s \in \mathbb{R}^{nc}}{\text{minimiser}} & \phi^S(x, \nu) = f(x) + \nu \sum_{i \in \mathcal{E}} (c_i(x) + 2s_i) + \nu \sum_{i \in \mathcal{I}} s_i \\ \text{s.c. :} & c_i(x) + s_i \geq 0 \quad \forall i \in \mathcal{C} = \mathcal{I} \cup \mathcal{E}, \\ & s_i \geq 0 \quad \forall i \in \mathcal{C}. \end{array} \tag{1.26}$$

On a alors les propriétés suivantes pour le problème (1.26) :

- Son domaine admissible est non-vide.
- La fonction objectif est différentiable en tout point du domaine admissible.
- La MFCQ est satisfaite en tout point du domaine admissible (voir (Gould et al., 2003, théorème 2.2)).

Remarquons que par le biais de cette transformation, on se retrouve avec un problème n'ayant que des contraintes d'inégalité.

Exemple 2

Soit le problème suivant où, pour simplifier l'explication, on choisit des contraintes

linéaires :

$$\begin{array}{ll} \text{minimiser} & x_1^2 + x_2^2 \\ \text{s.c. :} & x_1 = x_2, \\ & x_1 \geq 1. \end{array} \tag{1.27}$$

La reformulation de type (1.26) donne :

$$\begin{array}{ll} \text{minimiser} & x_1^2 + x_2^2 + \nu(x_1 - x_2 + 2s_1 + s_2) \\ \text{s.c. :} & x_1 - x_2 + s_1 \geq 0, \\ & x_1 - 1 + s_2 \geq 0, \\ & s_1 \geq 0 \text{ et } s_2 \geq 0. \end{array} \tag{1.28}$$

La figure 1.7 illustre la situation.

Remarque : On comprend alors mieux pourquoi PSUPERB a de bonnes chances de se comporter honorablement sur les problèmes de type MPEC (et donc sur les problèmes de structure). En effet, puisqu'il transforme le domaine admissible en l'élargissant, la figure 1.3 est remplacée par la figure 1.8.

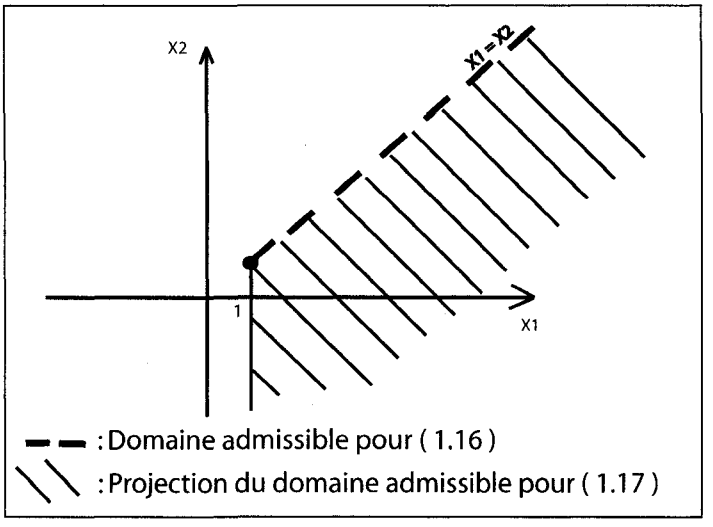


Figure 1.7 – Coupe du domaine admissible par l'espace $\{(x_1, x_2, s_1, s_2) \mid s_1 = s_2 = 0\}$ pour l'exemple 2

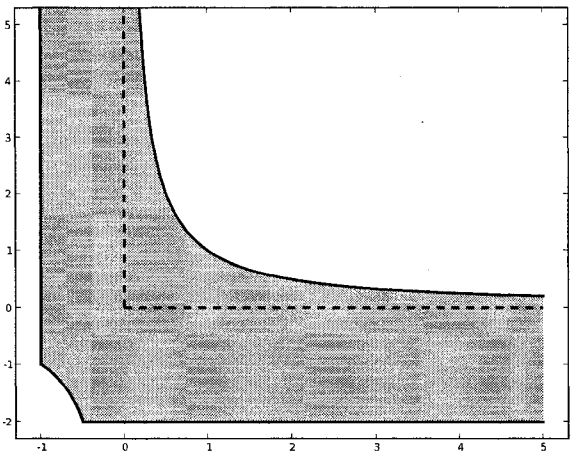


Figure 1.8 – Domaine admissible d'un MPEC après reformulation

1.2.5 Rappel sur l'algorithme au complet

Nous venons de voir que PSUPERB commence par reformuler le problème (1.25) sous la forme (1.26). On a alors affaire à un problème d'optimisation non-linéaire avec contraintes d'inégalité. La stratégie consiste alors à appliquer une méthode de points intérieurs.
Pour cela, on minimise la suite de fonctions barrières

$$\phi^B(x,s;\mu,\nu) \equiv \phi^S(x,s;\nu) - \mu \sum_{i\in\mathcal{C}} \log(c_i(x)+s_i) - \mu \sum_{i\in\mathcal{C}} s_i, \tag{1.29}$$

pour une suite de paramètres positifs $\{\mu^k\}$ décroissante dont la limite est zéro et une suite de paramètres de pénalisation positifs $\{\nu^k\}$ croissante.

Notation 1.2.5.1 *On note* $e_{\mathcal{E}} = (1...1) \in \mathbb{R}^{n_{\mathcal{E}}}$ *et* $e_{\mathcal{I}} = (1...1) \in \mathbb{R}^{n_{\mathcal{I}}}$.
Ensuite on donne les vecteurs $e^0_{\mathcal{E}} = \begin{bmatrix} e_{\mathcal{E}} \\ 0 \end{bmatrix}$, *et* $e^0_{\mathcal{I}} = \begin{bmatrix} 0 \\ e_{\mathcal{I}} \end{bmatrix}$ *de* $\mathbb{R}^{nc}$.
Finalement on pose $e = e^0_{\mathcal{E}} + e^0_{\mathcal{I}}$.

Notation 1.2.5.2 *Les matrices Jacobiennes de* $c_{\mathcal{E}}(x)$ *et de* $c_{\mathcal{I}}(x)$ *sont notées* $J_{\mathcal{E}}(x)$ *et* $J_{\mathcal{I}}(x)$ *respectivement et celle de* $c(x) = (c_{\mathcal{E}}(x), c_{\mathcal{I}}(x))$ *est* $J(x) = \begin{bmatrix} J_{\mathcal{E}}(x) \\ J_{\mathcal{I}}(x) \end{bmatrix}$.

Définition 1.2.5.1 *(Multiplicateurs de Lagrange primaux)*
Les multiplicateurs de Lagrange primaux sont donnés par (en utilisant les notations précédentes)

$$y(x,s) \equiv \mu(C(x)+S)^{-1}e, \tag{1.30}$$
$$u(s) \equiv \mu S^{-1}e. \tag{1.31}$$

Les multiplicateurs $y = (y_{\mathcal{E}}, y_{\mathcal{I}})$ sont associés aux contraintes $c = (c_{\mathcal{E}}, c_{\mathcal{I}})$ et les multiplicateurs u sont associés aux contraintes de bornes sur s.

Notation 1.2.5.3 *(Multiplicateurs décalés)*
On appelle multiplicateurs décalés les quantités suivantes (avec y des multiplicateurs primaux ou primaux-duaux)

$$\lambda(y,\nu) \equiv y - \nu e^0_{\mathcal{E}}. \tag{1.32}$$

Définition 1.2.5.2 *(Fonction d'arrêt)*
Une fonction d'arrêt $\epsilon(.)$ est une fonction réelle telle que $\epsilon(\mu) > 0$ pour tout $\mu > 0$ et $\epsilon(\mu) \downarrow 0$ quand $\mu \downarrow 0$.

Définition 1.2.5.3 *(Variables primale, duale et primale-duale)*
On définit la variable primale v par $v \equiv (x,s)$, la variable duale v_D par $v_D \equiv (y,u)$ et la variable primale-duale v_{PD} par $v_{PD} \equiv (x,s,y,u)$.

Notation 1.2.5.4 *On définit la fonction primale-duale $\Phi : \mathbb{R}^{n+3n_C} \to \mathbb{R}^{n+3n_C}$ par*

$$\Phi(v;\mu,\nu) \equiv \begin{bmatrix} \nabla f(x) - J^T(x)(y - \nu e^0_{\mathcal{E}}) \\ \nu e - (y - \nu e^0_{\mathcal{E}}) - u \\ (C(x)+S)y - \mu e \\ Su - \mu e \end{bmatrix}. \tag{1.33}$$

Les conditions d'optimalité primales-duales pour (1.29) s'écrivent

$$\Phi(v;\mu,\nu) = 0, \tag{1.34a}$$
$$\text{et } (c(x)+s, s, y, u) \geq 0. \tag{1.34b}$$

On sait aussi que les conditions de KKT pour (1.26) sont (1.34) avec (1.34a) remplacée par

$$\Phi(v; 0, \nu) \quad = \quad 0. \tag{1.35}$$

Définition 1.2.5.4 *(Règle Classique)*
Supposons qu'on soit à l'itération k de l'algorithme PSUPERB. Soit la règle :
Si $||c_{\mathcal{E}}(x^k)|| > \eta_{\mathcal{E}}^k$ ou $\max(-c_{\mathcal{I}}(x^k), 0) > \eta_{\mathcal{I}}^k$ alors poser $\nu^{k+1} = \max(\alpha\nu^k, \nu^k + \beta)$, où $\alpha > 1$, $\beta > 0$, $\eta_{\mathcal{E}}^k > 0$ et $\eta_{\mathcal{I}}^k > 0$ sont des paramètres.

La règle (1.2.5.4) est utilisée pour mettre à jour le paramètre de pénalité dans PSUPERB. On peut à présent énoncer l'algorithme complet *i.e.*, l'algorithme 1.2.5.

Remarque : Détaillons les mécanismes mis en jeu lors de l'étape 1 de l'algorithme 1.2.5, c'est-à-dire l'itération interne. A cette étape, on doit résoudre

$$\underset{(x,s)}{\text{minimiser}}\ \phi^B(x, s; \mu, \nu), \tag{1.36}$$

jusqu'à ce que les conditions d'arrêt (1.39) soient satisfaites. Notons que ϕ^B est définie en (1.29). Pour résoudre (1.36), on applique une méthode de région de confiance. Puisque PSUPERB est une méthode de points intérieurs primale-duale, cela signifie qu'on résout le problème (voir (Conn, Gould, & Toint, 2000))

$$\underset{d \in \mathcal{B}_k}{\text{minimiser}}\ \nabla_{v_P}\phi^B(x_k, s_k; \mu, \nu)^T d + \tfrac{1}{2} d^T H^{PD} d, \tag{1.37}$$

avec H^{PD} définie par

$$H^{PD} = \begin{bmatrix} H(x, \lambda(y, \nu)) + J^T(x)\Theta(v)J(x) & J^T(x)\Theta(v) \\ \Theta(v)J(x) & \Theta(v) + US^{-1} \end{bmatrix}, \tag{1.38}$$

avec (y, u) les variables primales-duales (d'où l'indice PD), $\lambda(y, \nu)$ définis en (1.2.5.3), J la Jacobienne des contraintes,

$$\Theta(v) = Y(C(x) + S)^{-1}$$

et où

$$H(x,\lambda) = \nabla_{xx} f(x) - \sum_{i \in \mathcal{C}} \lambda_i \nabla_{xx} c_i(x) = \nabla_{xx} L(x,\lambda)$$

est le Hessien du Lagrangien (voir définition 1.1.2.5). C'est ce modèle quadratique qu'on utilise dans PSUPERB lorsqu'on résout un sous-problème de région de confiance.

Algorithme 1.2.5 PSUPERB

Etape 0. Choisir des fonctions d'arrêt ϵ^D, ϵ^C et ϵ^U et un paramètre $\kappa_\nu > 0$. Choisir $x^0 \in \mathbb{R}^n$, $s^0 \in \mathbb{R}^{nc}_+$ tel que $c(x^0) + s^0 > 0$, des estimations y^0, $u^0 \in \mathbb{R}^{nc}_+$ et des paramètres de pénalité et de barrière ν^0 et $\mu^0 > 0$. Poser $k = 0$.

Etape 1. Itération interne : trouver un nouvel itéré primal-dual $v_{PD}^{k+1} = (x^{k+1}, s^{k+1}, y^{k+1}, u^{k+1})$ qui satisfait

$$\left\| \begin{bmatrix} \nabla f(x^{k+1}) - J^T(x^{k+1})(y^{k+1} - \nu^k e_{\mathcal{E}}^0) \\ \nu^k e - (y^{k+1} - \nu^k e_{\mathcal{E}}^0) - u^{k+1} \end{bmatrix} \right\|_{[P^{k+1}]} \leq \epsilon^D(\mu^k), \quad (1.39a)$$

$$\|(C(x^{k+1}) + S^{k+1})y^{k+1} - \mu^k e\| \leq \epsilon^C(\mu^k), \quad (1.39b)$$

$$\|S^{k+1}u^{k+1} - \mu^k e\| \leq \epsilon^U(\mu^k), \quad (1.39c)$$

$$(c(x^{k+1}) + s^{k+1}, s^{k+1}) > 0, \quad (1.39d)$$

$$(\nu^k[e + e_{\mathcal{E}}^0] + \kappa_\nu e, \nu^k[e + e_{\mathcal{E}}^0] + \kappa_\nu e) > (y^{k+1}, u^{k+1}) > 0. \quad (1.39e)$$

pour une norme de mise à l'échelle $||.||_{[P^{k+1}]}$ adaptée.

Etape 2. Choisir un nouveau paramètre barrière $\mu^{k+1} \in [0, \mu^k[$ tel que $\lim_{k \to \infty} \mu^k = 0$. Si nécessaire, ajuster le paramètre ν^k. Incrémenter k de 1 et retourner à l'étape 1.

Pour terminer, on peut énoncer deux résultats de convergence de PSUPERB présentés dans (Gould et al., 2003) :

Théorème 1.2.2 *(Convergence de l'itération interne) Appelons itération interne la résolution de (1.37) jusqu'à ce que les conditions (1.39) soient satisfaites. Alors, sous certaines hypothèses standards en méthodes de points intérieurs, la procédure d'itération interne correspondant à l'itération externe k de l'algorithme 1.2.5 génère une suite $\{(v_P^k, v_D^k)\}$ qui satisfait les conditions d'arrêt de l'étape 1 après un nombre fini de pas de calcul.*

Nous donnons maintenant quelques hypothèses qui nous seront utiles par la suite.

H1 Le domaine admissible pour (NLP) est non vide.

H2 $f, c_{\mathcal{E}}$ et $c_{\mathcal{I}}$ sont de classe $\mathcal{C}^2$ dans un voisinage contenant les itérés générés par l'algorithme 1.2.5 et les gradients et Hessiens de ces fonctions sont uniformément bornés sur ce voisinage.

H2 Bis $f, c_{\mathcal{E}}$ et $c_{\mathcal{I}}$ sont de classe $\mathcal{C}^1$ dans un voisinage contenant les itérés générés par l'algorithme 1.2.5 et les gradients et Hessiens de ces fonctions sont uniformément bornés sur ce voisinage.

H3 L'algorithme 1.2.5 génère une suite $\{x^k, s^k, y^k, u^k\}$ infinie et le paramètre ν est mis à jour un nombre fini de fois jusqu'à atteindre sa valeur finale ν^*.

H4 L'algorithme 1.2.5 génère une suite $\{x^k, s^k, y^k, u^k\}$ infinie et le paramètre ν est mis à jour un nombre infini de fois.

Théorème 1.2.3 *(Convergence de l'itération externe - Cas où ν reste borné) Supposons H1, H2 et H3. Alors la suite $\{s^k, y^k, u^k\}$ est bornée. De plus, si $\{x^k\}$ a un point d'adhérence, notons $\{x^*, s^*, y^*, u^*\}$ un point d'adhérence de $\{x^k, s^k, y^k, u^k\}$. Alors $s^* = 0$ et le point $\{x^*, \lambda(y^*, \nu^*)\}$ est un point critique du premier ordre pour (NLP).*

Théorème 1.2.4 *(Convergence de l'itération externe - Cas où ν n'est pas borné) Supposons H1, H2 et H4. Alors la suite $\{y^k, u^k\}$ est non bornée et tout point d'adhérence de $\{x_k\}$ minimise la violation des contraintes.*

1.2.6 Théorèmes relatifs à PSUPERB

1.2.6.1 PSUPERB et la MFCQ

Nous établissons dans ce qui suit un résultat concernant le comportement de PSUPERB lorsque la MFCQ n'est pas satisfaite au point optimal. Ce théorème constitue une sorte de certificat de dégénérescence au point optimal.

Définition 1.2.6.1 *(Règle Spéciale)*
Soit la règle :
Si $||y^k - \nu^k e^0_{\mathcal{E}}|| > \gamma\nu^k$ alors poser $\nu^{k+1} = \max(\alpha\nu^k, \nu^k + \beta)$, où $\gamma \in]0,1[$, $\alpha > 1$ et $\beta > 0$.

Dans l'algorithme, on augmente ν si au moins une des deux règles 1.2.5.4 ou 1.2.6.1 est vérifiée.

Théorème 1.2.5 *(PSUPERB et la MFCQ) Supposons H2 Bis, H4 et $\{x_k, s_k\}$ possède un point d'adhérence $\{x^*, s^*\}$ tel que $\{x^*\}$ est admissible. On a alors :*

1) $s^* = 0$.

2) *x^* est un minimum global pour la violation des contraintes.*

De plus, si on utilise les règles de mise à jour définies en 1.2.5.4 et 1.2.6.1 et que cette dernière est responsable de la convergence de $\{\nu^k\}$ vers l'infini alors on peut conclure que :

3) *x^* est un point de Fritz-John pour (1.1) avec $\lambda_0^* = 0$, où λ_0^* correspond à la notation du théorème (1.1.6).*

4) *La MFCQ n'est pas satisfaite en x^*.*

Preuve :

1) Soit la première mesure d'inadmissiblité :

$$\vartheta_P(x) = \sum_{i \in \mathcal{E}} |c_i(x)| + \sum_{i \in I} \max[-c_i(x), 0]$$

Il est clair que puisque x^* est admissible pour (1.1), alors $\vartheta_P(x^*) = 0$.
On a toujours : $c_i(x) = r_i - s_i$ où r_i est la partie positive et s_i la partie négative de $c_i(x)$.
D'où on tire :

$$\begin{aligned} |c_i(x)| &= r_i + s_i \\ \text{et donc} \quad |c_i(x)| &= c_i(x) + 2s_i. \end{aligned}$$

On a aussi :

$$\max[-c_i(x), 0] = s_i.$$

Puisque $\vartheta_P(x^*) = 0$ on a :

$$\begin{aligned} |c_i(x^*)| &= 0, \quad \forall i \in \mathcal{E} \\ \text{et} \quad \max[-c_i(x^*), 0] &= 0, \quad \forall i \in I. \end{aligned}$$

Donc

$$s_i^* = r_i^* = 0, \quad \forall i \in \mathcal{E},$$
$$s_i^* = 0, \quad \forall i \in I.$$

Finalement $s^* = 0$.

2) La mesure d'inadmissibilité est positive ou nulle. Or quand x^* est admissible elle est nulle. Donc avec les hypothèses du théorème on a

$$\vartheta(x^*, s^*) = 0$$

donc x^* est un minimum global pour la violation des contraintes.

3) Supposons maintenant que ν^k est mis à jour un nombre infini de fois mais seulement un nombe fini de fois à cause de la règle 1.2.5.4. On pose alors

$$\zeta^{k+1} = \max\{||y_{\mathcal{E}}^{k+1} - \nu^k e_{\mathcal{E}}||_\infty, ||y_{\mathcal{I}}^{k+1}||_\infty\}.$$

Prenons la première ligne de l'équation (1.39a) et divisons de part et d'autre par ζ^{k+1}. On obtient, après utilisation de (Gould et al., 2003, Lemme 4.3)

$$||\frac{1}{\zeta^{k+1}}\nabla f(x^{k+1}) - J^T(x^{k+1})\frac{(y^{k+1} - \nu^k e_{\mathcal{E}}^0)}{\zeta^{k+1}}|| \leq \frac{1}{\zeta^{k+1}}\epsilon^D(\mu^k). \tag{1.40}$$

Posons à présent $\overline{y}^{k+1} = (y^{k+1} - \nu^k e_{\mathcal{E}}^0)/\zeta^{k+1}$. Par construction, $||\overline{y}^{k+1}||_\infty = 1$ pour tout k, donc la suite des $\{\bar{y}^{k+1}\}$ est bornée. Soit $\overline{y}^*$ un point d'adhérence de la suite $\{\bar{y}^k\}$. On a ainsi $||\overline{y}^*||_\infty = 1$ et donc $\overline{y}^*$ possède au moins une composante non nulle. De plus on a $\lim_{k\to\infty} \zeta^k = +\infty$ d'après l'hypothèse sur les causes de la mise à jour de ν^k. Donc après passage à la limite, (1.40) devient alors

$$J^T(x^*)\overline{y}^* = 0, \tag{1.41}$$

avec $\overline{y}^*$ qui n'est pas le vecteur nul. x^* est donc un point de Fritz-John avec $\lambda_0^* = 0$ si on reprend les notations du théorème (1.1.6).

4) On sait que la MFCQ est équivalente à la BCQ (voir définition 1.1.2.2).

Or, on vient de montrer dans **3)** que

$$\exists \overline{y}^* \text{tq } \overline{y}^* \neq 0 \text{ et } J^T(x^*)\overline{y}^* = 0$$

Donc la BCQ est violée en x^* et donc la MFCQ n'est pas satisfaite en x^* d'après (1.2). □

1.2.6.2 PSUPERB et les MPVC

Il faut savoir que la classe de problèmes MPVC a vu le jour récemment puisqu'elle a été introduite dans l'article (Achtziger & Kanzow, 2008). Elle repose sur l'exploration des problèmes de structure mais on peut facilement imaginer que dans un futur proche, d'autres domaines d'application auront recours à des formulations de type (MPVC). Nous présentons ici des résultats concernant la qualité des points optimaux obtenus grâce à PSUPERB sur des problèmes de type (MPVC). Nous introduisons d'abord des conditions de qualification adaptées aux (MPVC) et ensuite nous formulerons un théorème qui s'apparente au théorème 1.2.5.

La premère condition de qualification adaptée aux MPVC est présentée dans (Achtziger & Kanzow, 2008). Cette condition ressemble à la MFCQ (voir 1.1.2.3).

Définition 1.2.6.2 *(VC-MFCQ)*
Soit x^* une solution de (MPVC). On dit alors que la VC-MFCQ est satisfaite en x^* si et seulement si les gradients

$$\{\nabla h_i(x^*)\}_{i\in I_h} \cup \{\nabla H_i(x^*)\}_{i\in I_{00}\cup I_{0+}}$$

sont linéairement indépendants et s'il existe une direction d telle que

$$\nabla h_i^T(x^*)d = 0, \qquad \forall i \in I_h, \tag{1.42a}$$
$$\nabla H_i^T(x^*)d = 0, \qquad \forall i \in I_{00} \cup I_{0+}, \tag{1.42b}$$
$$\nabla g_i^T(x^*)d < 0, \qquad \forall i \in I_{g0}, \tag{1.42c}$$
$$\nabla G_i^T(x^*)d < 0, \qquad \forall i \in I_{+0}, \tag{1.42d}$$
$$\nabla H_i^T(x^*)d > 0, \qquad \forall i \in I_{0-}, \tag{1.42e}$$

où $I_{00} = \{i \in 1,...,l \mid H_i(x^*) = G_i(x^*) = 0\}$, $I_{0+} = \{i \in 1,...,l \mid H_i(x^*) = 0$ *et* $G_i(x^*) > 0\}$, $I_{0-} = \{i \in 1,...,l \mid H_i(x^*) = 0$ *et* $G_i(x^*) < 0\}$, $I_{+0} = \{i \in 1,...,l \mid H_i(x^*) > 0$ *et* $G_i(x^*) = 0\}$ *et* $I_{g0} = \{i \in I_g \mid g_i(x^*) = 0\}$.

De même, nous définissons une VC-BCQ, similaire à celle énoncée en 1.1.2.2.

Définition 1.2.6.3 *(VC-BCQ)*
Soit $(\omega, \lambda) \in \mathbb{R}^{|I_h|+|I_{00}\cup I_{0+}|} \times \mathbb{R}_+^{|I_{g0}|+|I_{+0}|+|I_{0-}|}$ *et* x^* *une solution de (MPVC). On dit alors que la VC-BCQ est satisfaite en* x^* *si et seulement si*

$$\begin{array}{l} \sum_{i\in I_h} \omega_i \nabla h_i(x^*) + \sum_{j\in I_{00}\cup I_{0+}} \omega_j \nabla H_j(x^*) \\ + \sum_{k\in I_{g0}} \lambda_k \nabla g_k(x^*) + \sum_{q\in I_{+0}} \lambda_q \nabla G_q(x^*) \implies (\omega, \lambda) = (0,0). \\ - \sum_{m\in I_{0-}} \lambda_m \nabla H_m(x^*) = 0 \end{array}$$

Le théorème 1.2.6 permet d'établir un lien entre ces deux conditions de qualification.

Théorème 1.2.6 *Soit* x^* *une solution de (MPVC). Si la VC-MFCQ est satisfaite en* x^* *alors la VC-BCQ l'est aussi.*

Preuve : Soit x^* une solution de (MPVC). Supposons que la VC-MFCQ est satisfaite en x^*. Posons

$$A^T = \begin{pmatrix} \nabla g_k^T(x^*), & & \text{pour } k \in I_{g0} \\ & \cdots & \\ \nabla G_q^T(x^*), & & \text{pour } q \in I_{+0} \\ & \cdots & \\ -\nabla H_m^T(x^*), & & \text{pour } m \in I_{0-} \end{pmatrix}$$

et

$$C^T = \begin{pmatrix} \nabla h_i^T(x^*), & & \text{pour } i \in I_h \\ & \cdots & \\ \nabla H_j^T(x^*), & & \text{pour } j \in I_{00} \cup I_{0+} \end{pmatrix}.$$

Puisque la VC-MFCQ est, par hypothèse, satisfaite en x^*, il existe un d tel que $A^T d < 0$ et $C^T d = 0$. D'après le théorème de Motzkin (voir (Mangasarian, 1994)) on peut affirmer qu'il est impossible de satisfaire

$$Az_1 + Cz_3 = 0 \text{ avec } z_1 \geq 0 \text{ et } z_1 \neq 0. \tag{1.43}$$

Prenons à présent $(\omega, \lambda) \in \mathbb{R}^{|I_h|+|I_{00}\cup I_{0+}|} \times \mathbb{R}_+^{|I_{g0}|+|I_{+0}|+|I_{0-}|}$ et supposons que

$$\sum_{i\in I_h} \omega_i \nabla h_i(x^*) + \sum_{j\in I_{00}\cup I_{0+}} \omega_j \nabla H_j(x^*) + \sum_{k\in I_{g0}} \lambda_k \nabla g_k(x^*) + \sum_{q\in I_{+0}} \lambda_q \nabla G_q(x^*) - \sum_{m\in I_{0-}} \lambda_m \nabla H_m(x^*) = 0.$$

On peut appliquer (1.43) avec $\lambda \equiv z_1$ et $\omega \equiv z_3$. On en conclut que $\lambda = 0$. Ensuite puisque $\{\nabla h_i(x^*)\}_{i\in I_h} \cup \{\nabla H_i(x^*)\}_{i\in I_{00}\cup I_{0+}}$ sont linéairement indépendants on a $\omega = 0$. Par conséquent, la VC-BCQ est satisfaite en x^*. □

Remarque : Étudions le lien qu'il y a entre la violation de la BCQ et de la VC-BCQ. Si en un point x^* la BCQ est violée cela signifie qu'on peut trouver des multiplicateurs $(\omega, \lambda, \psi) \neq (0, 0, 0)$ avec $\omega \in \mathbb{R}^{|I_h|}$, $\lambda \in \mathbb{R}_+^{|I_{g0}\cup I_0|}$ et $\psi \in \mathbb{R}_+^{|I_0\cup I_{+0}|}$ tels que

$$\sum_{i\in I_h} \omega_i \nabla h_i(x^*) + \sum_{j\in I_{g0}} \lambda_j \nabla g_j(x^*) - \sum_{q\in I_0} \lambda_q \nabla H_q(x^*) + \sum_{m\in I_0\cup I_{+0}} \psi_m \nabla(H_m.G_m)(x^*) = 0,$$

où $I_0 = I_{00} \cup I_{0+} \cup I_{0-}$. Lorsqu'on développe le terme $\nabla(H_m.G_m)(x^*)$ on obtient

$$\begin{aligned} &\textstyle\sum_{i\in I_h} \omega_i \nabla h_i(x^*) + \sum_{j\in I_{g0}} \lambda_j \nabla g_j(x^*) \\ &\quad + \textstyle\sum_{q\in I_0} (\psi_q G_q(x^*) - \lambda_q) \nabla H_q(x^*) \\ &\quad\quad + \textstyle\sum_{m\in I_{+0}} \psi_m H_m(x^*) \nabla G_m(x^*) \;=\; 0. \end{aligned} \tag{1.44}$$

La VC-BCQ est alors violée en x^* si

$$\begin{aligned} \omega_i &= 0, && \forall i \in I_h, \\ \lambda_j &= 0, && \forall j \in I_{g0}, \\ \psi_q G_q(x^*) - \lambda_q &= 0, && \forall q \in I_0, \\ \psi_m H_m(x^*) &= 0, && \forall m \in I_{+0}, \end{aligned}$$

n'est **pas** l'unique solution satisfaisant (1.44) avec les contraintes de signe suivantes :

$$\begin{aligned} \lambda_j &\geq 0, && \forall j \in I_{g0}, \\ \psi_q G_q(x^*) - \lambda_q &\leq 0, && \forall q \in I_{0-}, \\ \psi_m H_m(x^*) &\geq 0, && \forall m \in I_{+0}. \end{aligned} \tag{1.45}$$

On peut maintenant énoncer un résultat similaire au théorème 1.2.5 qui permet, dans certains cas, d'obtenir un certificat de violation de la VC-MFCQ. Avant cela, il convient de faire un point sur les diverses notations utilisées. En faisant l'analogie avec les multiplicateurs introduits dans la section 1.2.5 on a

$$\begin{array}{rcl} y_{\mathcal{E}} & \equiv & \omega \\ & & \left(\begin{array}{ll} \lambda_j, & \text{pour } j \in I_g \\ \quad \cdots & \\ \lambda_q, & \text{pour } q \in 1, ..., l \\ \quad \cdots & \\ \psi_m, & \text{pour } m \in 1, ..., l \end{array} \right) . \\ y_{\mathcal{I}} & \equiv & \end{array} \tag{1.46}$$

On peut alors énoncer une nouvelle règle de mise à jour du paramètre de pénalité.

Définition 1.2.6.4 *(Règle Spéciale MPVC)*
Soit la règle :
Si $||(\omega_i^{k+1} - \nu^k)_{i \in I_h}, (\lambda_i^{k+1})_{i \in I_g}, (\psi_i^{k+1} H_i(x^{k+1}))_{i \in 1,...,l}, (\psi_i^{k+1} G_i(x^{k+1}) - \lambda_i^{k+1})_{i \in 1,...,l}||_\infty > \gamma \nu^k$ *alors poser* $\nu^{k+1} = \max(\alpha \nu^k, \nu^k + \beta)$, *où* $\gamma \in]0, 1[$, $\alpha > 1$ *et* $\beta > 0$.

Théorème 1.2.7 *(PSUPERB et la VC-MFCQ) Supposons H4 et* $\{x_k, s_k\}$ *possède un point d'adhérence* (x^*, s^*) *tel que* x^* *est admissible pour (MPVC). On utilise les règles de mise à jour définie en 1.2.5.4 et 1.2.6.4. On suppose qu'on met à jour* ν *un nombre infini de fois et qu'on utilise un nombre fini de fois la règle 1.2.6.4 pour augmenter* ν. *On peut alors conclure que la VC-MFCQ est violée en* x^*.

Preuve : On reprend l'idée de la preuve de 1.2.5. Supposons que l'algorithme converge vers x^* admissible pour (MPVC) et H4. On suppose aussi que ν^k est mis à jour une infinité de fois à cause de la règle définie en 1.2.6.4. On pose

$$\begin{array}{rl} \zeta^{k+1} = & \max\{||(\omega_i^{k+1} - \nu^k)_{i \in I_h}||_\infty, ||(\lambda_i^{k+1})_{i \in I_g}||_\infty, \\ & ||(\psi_i^{k+1} H_i(x^{k+1}))_{i \in 1,...,l}||_\infty, ||(\psi_i^{k+1} G_i(x^{k+1}) - \lambda_i^{k+1})_{i \in 1,...,l}||_\infty\}. \end{array}$$

On divise la première ligne de l'équation (1.39a) de part et d'autre par ζ^{k+1} et on

obtient (après utilisation de (Gould et al., 2003, Lemme 4.3))

$$\left\| \frac{1}{\zeta^{k+1}} \begin{pmatrix} \nabla f(x^{k+1}) - \sum_{i \in I_h} (\omega_i^{k+1} - \nu^k) \nabla h_i(x^{k+1}) + \sum_{j \in I_g} \lambda_j^{k+1} \nabla g_j(x^{k+1}) \\ -\sum_{q=1}^{l} \lambda_q^{k+1} \nabla H_q(x^{k+1}) + \sum_{m=1}^{l} \psi_m^{k+1} \nabla (H_m.G_m)(x^{k+1}) \end{pmatrix} \right\| \tag{1.47}$$

$$\leq \frac{\epsilon^D(\mu^k)}{\zeta^{k+1}}.$$

Puisqu'on suppose que x^* est admissible alors $s^* = 0$ (voir théorème 1.2.5). Par (1.39b) et (1.39c) on sait qu'il ne reste que les termes

$$\begin{aligned} &\omega_i, \text{ pour } i \in I_h, && \lambda_j, \text{ pour } j \in I_{g0}, \\ &\lambda_q, \text{ pour } q \in I_0, && \psi_m, \text{ pour } m \in I_0 \cup I_{0+}, \end{aligned}$$

car tous les autres convergent vers 0. En ne retenant que les termes dominants, l'équation (1.47) devient alors

$$\lim_{k \to \infty} \left\| \frac{1}{\zeta^{k+1}} \begin{pmatrix} \nabla f(x^{k+1}) - \sum_{i \in I_h} (\omega_i^{k+1} - \nu^k) \nabla h_i(x^{k+1}) + \sum_{j \in I_{g0}} \lambda_j^{k+1} \nabla g_j(x^{k+1}) \\ -\sum_{q \in I_0} \lambda_q^{k+1} \nabla H_q(x^{k+1}) + \sum_{m \in I_0 \cup I_{+0}} \psi_m^{k+1} \nabla (H_m.G_m)(x^{k+1}) \end{pmatrix} \right\|$$

$$= 0. \tag{1.48}$$

D'après l'hypothèse sur les causes de la mise à jour de ν^k on a $\lim_{k \to \infty} \zeta^k = +\infty$. On pose maintenant

$$\begin{aligned} \overline{\omega}_i^{k+1} &\equiv (\omega_i^{k+1} - \nu^k e_i)/\zeta^{k+1}, \quad \forall i \in I_h, \\ \overline{\lambda}_j^{k+1} &\equiv \lambda_j^{k+1}/\zeta^{k+1}, \quad \forall j \in I_g, \\ \bar{\theta}_q^{k+1} &\equiv (\psi_q^{k+1} G_q(x_{k+1}) - \lambda_q^{k+1})/\zeta^{k+1}, \quad \forall q \in 1, ..., l, \\ \bar{\phi}_m^{k+1} &\equiv \psi_m^{k+1} H_m(x_{k+1})/\zeta^{k+1}, \quad \forall m \in 1, ..., l. \end{aligned}$$

Par construction $||(\overline{\omega}^{k+1}, \overline{\lambda}^{k+1}, \bar{\theta}^{k+1}, \bar{\phi}^{k+1})||_\infty = 1$. Soit

$$(\overline{\omega}^*, \overline{\lambda}^*, \bar{\theta}^*, \bar{\phi}^*)$$

un point d'adhérence de cette suite. Alors $||(\overline{\omega}^*, \overline{\lambda}^*, \bar{\theta}^*, \bar{\phi}^*)||_\infty = 1$ et donc ses com-

posantes ne sont pas toutes nulles. Donc après passage à la limite dans (1.48) et en développant directement le terme $\nabla(H.G)(x^*)$ on obtient :

$$\begin{aligned}&\sum_{i\in I_h}\overline{\omega}_i^*\nabla h_i(x^*)+\sum_{j\in I_{g0}}\overline{\lambda_j}^*\nabla g_j(x^*)\\&+\sum_{q\in I_0}(\overline{\psi}_q^*G_q(x^*)-\overline{\lambda}_q^*)\nabla H_q(x^*)+\sum_{m\in I_{+0}}\overline{\psi}_m^*H_m(x^*)\nabla G_m(x^*)\quad=0.\end{aligned}\tag{1.49}$$

On sait que les composantes de $(\overline{\mu}^*,\overline{\lambda}^*,\bar{\theta}^*,\bar{\phi}^*)$ sont non toutes nulles. Il reste à montrer que ses composantes satisfont les contraintes de signe (1.45) et nous aurons alors montré que la VC-BCQ est violée en x^*.

- On sait, d'après les propriétés de l'algorithme, que $\lambda^* \geq 0$ donc *a fortiori* $\lambda_j^* \geq 0$, pour tout $j \in I_{g0}$. Finalement $\forall j \in I_{g0}, \overline{\lambda}_j^* \geq 0$ (car $\forall k, \zeta^k \geq 0$).
- Pour $q \in I_{0-}$ on a $G_q(x^*) < 0$ donc $\overline{\psi}_q^* G_q(x^*) \leq 0$ et finalement on a bien $\overline{\psi}_q^* \overline{G}_q(x^*) - \overline{\lambda}_q^* \leq 0$ (puisque $\overline{\lambda}^* \geq 0$) c'est-à-dire $\bar{\theta}_q^* \leq 0$.
- Pour $m \in I_{+0}$ on a $H_m(x^*) > 0$ donc $\overline{\psi}_m^* H_m(x^*) \geq 0$ et finalement on a bien $\bar{\phi}_m^* \geq 0$.

Les contraintes de signes sont bien respectées et la VC-BCQ est donc violée. Par la contraposée du théorème 1.2.6 on a alors que la VC-MFCQ est violée en x^*. □

1.3 Classe de problèmes considérés

Les problèmes que nous souhaitons étudier sont des problèmes de structure. De tels problèmes peuvent être rencontrés pratiquement, par exemple, dans le domaine de la construction. Ainsi, on peut garder en tête l'image d'un pont à structure métallique dont on chercherait à minimiser le poids tout en s'assurant qu'il est capable de supporter les contraintes de résistance mentionnées dans le cahier des charges. Dans notre étude, nous considérons une structure plus modeste, en deux dimensions, composée de barres rigides reliées entre elles par des noeuds. On appelle cette structure planaire un *treillis*. Ces barres peuvent alors être soumises à des forces extérieures (qu'on suppose constantes) qui entraînent des déformations du treillis. La figure 1.9 illustre ce cas.

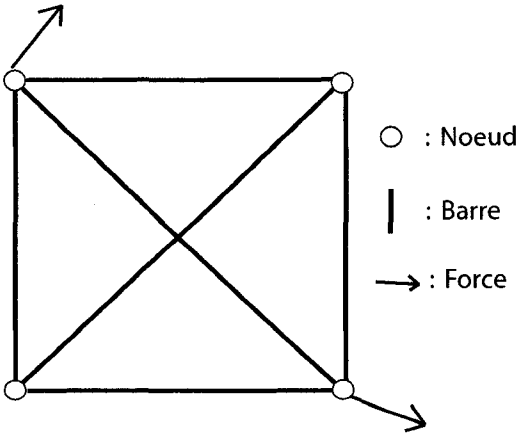


Figure 1.9 – Treillis de barres soumis à des forces extérieures

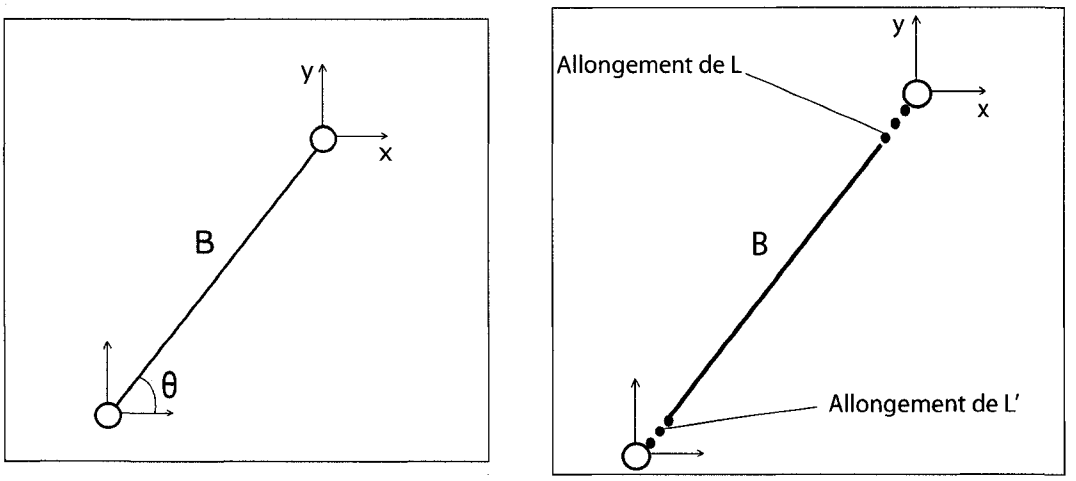
Lorsqu'on étudie un treillis, on s'aperçoit qu'il y a essentiellement deux variables qu'on peut ajuster :

(a) le diamètre de chaque barre qu'on notera a par la suite,

(b) le déplacement associé à chaque barre, noté u dans la suite.

Si la notion de diamètre de barre est intuitive, celle de déplacement nécessite une explication.

Soit une barre B dont les extrémités peuvent bouger selon les directions x et y (voir figure 1.10) et qui fait un angle θ avec l'horizontale.



(a) Barre du treillis et directions associées

(b) Barre B allongée

Figure 1.10 – Barres du treillis et élongation

La barre B a quatre degrés de liberté : deux pour l'extrémité du haut (un degré selon x et un selon y) et deux pour celle du bas (même explication). On associe alors à la barre un vecteur déplacement, u, à quatre composantes. Si la barre B s'allonge comme dans la figure 1.10 (b), le déplacement est alors

$$u = \begin{pmatrix} L\cos\theta \\ L\sin\theta \\ -L'\cos\theta \\ -L'\sin\theta \end{pmatrix}.$$

Il faut préciser qu'on ne considère pas θ comme une variable. Seules L et L' peuvent varier au cours du problème. Remarquons que l'allongement total de la barre vaut $A = L + L'$ (où A, L et L' sont algébriques). On définit alors le vecteur r (de même dimension que u) associé à la barre B et tel que

$$r^T u = A.$$

Ainsi, dans notre exemple, on a

$$r = \begin{pmatrix} \cos\theta \\ \sin\theta \\ -\cos\theta \\ -\sin\theta \end{pmatrix}.$$

D'après ce qu'on a écrit précédemment sur θ, r est donc fixé.

À présent, toutes les grandeurs utiles à la compréhension générale ont été interprétées physiquement. Nous pouvons donc poursuivre l'étude du problème. Il s'agit de modéliser la situation en termes d'objectif et de contraintes. Il existe en fait divers modèles pour représenter ces problèmes pratiques. Nous en donnons maintenant deux.

1.3.1 L'approche du volume minimal

1.3.1.1 Formulation classique

Dans cette section, nous nous concentrons sur une approche qui consiste à minimiser le volume de la structure. La formulation classique du problème consiste à écrire :

$$\begin{aligned} \underset{a}{\text{minimiser}} \quad & \sum_i l_i \rho_i a_i \\ \text{s.c. :} \quad & \sigma_i^{min} \leq \sigma_i(a) \leq \sigma_i^{max} \quad \text{si } a_i > 0, \\ & \sigma_i(a) \geq \sigma_i^{Cr}(a_i) \quad \text{si } a_i > 0, \\ & a \geq 0, \end{aligned} \tag{1.50}$$

où (pour la i-ème barre) on a l_i la longueur, ρ_i la densité du matériau, a_i le diamètre, σ_i^{min} (respectivement σ_i^{max}) l'effort maximal en compression (respectivement en tension), σ_i^{Cr} est l'effort de fissuration d'Euler : il représente l'effort maximum qu'on peut exercer au milieu de la barre avant de la fissurer et dépend de la géométrie de la barre [1] . Pour une barre à section circulaire on a $\sigma_i^{Cr} = -\pi E_i a_i/(4l_i^2)$ où E_i est le module de Young. On a aussi $\sigma_i(a) = (E_i/l_i) r_i^T u$, où u et r_i (*i.e.*, le vecteur r associé à la barre i) sont les vecteurs définis précédemment. Le vecteur déplacement doit vérifier

[1] En anglais, les contraintes associées à σ_i^{Cr} sont appelées *buckling constraints.*

$K(a)u = p$ où p est le vecteur des forces extérieures appliquées à la structure et K(a) la matrice de rigidité de la strucutre, définie par : $K(a) = \sum_{i=1}^{m} a_i(E_i/l_i)r_i r_i^T$, avec m le nombre de barres de la structure.

La formulation (1.50), bien que classique, n'est pas très commode. En effet, les contraintes sont difficiles à manipuler à cause de la condition «si $a_i > 0$». Par exemple, lorsqu'on veut utiliser un algorithme pour traiter ces problèmes, il faut préciser à l'algorithme quelles sont les contraintes qui existent mais comme cela dépend des variables, cela peut notablement compliquer les opérations. On préfère être en présence de contraintes qui sont toujours valables. On peut alors montrer (Achtziger, 1999) qu'il est équivalent de résoudre le problème suivant :

$$\underset{a,f,u}{\text{minimiser}} \quad \sum_{i=1}^{m} l_i \rho_i a_i \tag{1.51a}$$

$$\text{s.c. :} \quad \sum_{i=1}^{m} r_i f_i = p, \tag{1.51b}$$

$$f_i = a_i(E_i/l_i)r_i^T u \qquad \forall i = 1, ..., m, \tag{1.51c}$$

$$a_i \sigma_i^{min} \leq f_i \leq a_i \sigma_i^{max} \qquad \forall i = 1, ..., m, \tag{1.51d}$$

$$f_i \geq a_i \sigma_i^{Cr}(a_i) \qquad \forall i = 1, ..., m, \tag{1.51e}$$

$$a_i \geq 0 \qquad \forall i = 1, ..., m, \tag{1.51f}$$

où les notations sont les mêmes que pour la formulation (1.50). On s'est ainsi débarassé des contraintes difficiles à contrôler. De manière générale, face à cette formulation, il est assez difficile d'extraire des informations du point de vue de l'optimisation. En plus de l'évidente non-linéarité du problème (contrainte (1.51c) par exemple) on peut énoncer la propriété suivante.

Propriété. Soit $x^* = (a^*, f^*, u^*)$ un point optimal pour (1.51). Si l'une des barres du treillis a un diamètre nul, autement dit si $\exists i \in \{1, ..., m\}$ tel que $a_i^* = 0$ alors la LICQ est violée en x^*.

Preuve : Les contraintes

$$\begin{aligned} a_i^* &\geq 0, \\ f_i^* &= a_i^*(E_i/l_i)r_i^T u^*, \\ f_i^* &\leq a_i^* \sigma_i^{max}, \end{aligned}$$

sont actives puisque $a_i^* = 0$. Ordonnons les variables dans le sens a, f, u. Les gradients de ces contraintes sont alors

$$\begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ - \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ - \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ -(E_i/l_i)r_i^T u^* \\ \vdots \\ 0 \\ - \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ - \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad et \quad \begin{pmatrix} 0 \\ \vdots \\ \sigma_i^{max} \\ \vdots \\ 0 \\ - \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ - \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} \\ \\ \longleftarrow \partial/\partial a_i \\ \\ \\ \\ \\ \\ \longleftarrow \partial/\partial f_i \\ \\ \\ \\ \\ \\ \longleftarrow \partial/\partial u_i \\ \\ \end{matrix} .$$

Il est alors clair que ces trois vecteurs sont linéairement dépendants donc la LICQ est violée en x^*.□

Remarque : Dans les cas pratiques, il est très courant qu'une des barres «disparaisse» à l'optimalité et donc que la LICQ soit violée comme nous le constaterons dans le chapitre 3.

Exemple

On considère l'exemple académique, proposé dans (Guo, Cheng, & Yamazaki, 2001) du treillis à trois barres, soumis à la force extérieure verticale P = -30 et avec les deux directions x et y précisées sur la figure 1.11.

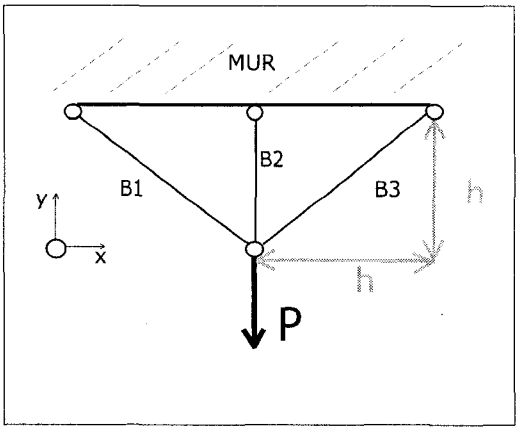


Figure 1.11 – Treillis à trois barres

On prend les valeurs suivantes des paramètres :

$$\sigma_i^{max} = -\sigma_i^{min} = 5,\ l_i = 1 \text{ et } E_i = 1,\ \text{pour } i \in \{1, 2, 3\}.$$

On a aussi

$$r_1 = \begin{pmatrix} \sqrt{2}/2 \\ -\sqrt{2}/2 \end{pmatrix},\ r_2 = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \text{ et } r_3 = \begin{pmatrix} -\sqrt{2}/2 \\ -\sqrt{2}/2 \end{pmatrix}.$$

Dans ces conditions la solution théorique au problème 1.51, donnée dans (Guo et al., 2001) est illustrée par la figure 1.12. D'après la propriété précédente, on sait donc que pour cet exemple, la LICQ n'est pas satisfaite au point optimal. Soulignons que la plupart des codes d'optimisation non-linéaire actuellement disponibles suppose que la LICQ est vérifiée au point optimal.

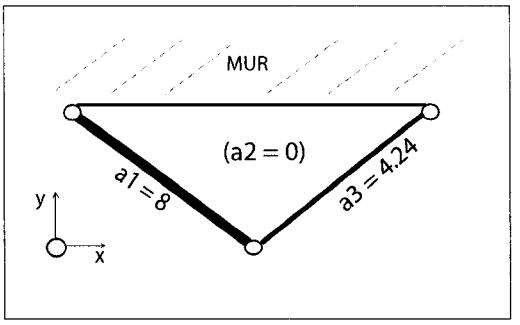


Figure 1.12 – Treillis à trois barres optimal

1.3.1.2 Formulation MPVC

Le problème de structure peut aussi s'écrire comme un MPVC (voir (MPVC)). Cela revient à transformer (1.51) de la façon suivante :

$$\underset{a,f,u}{\text{minimiser}} \quad \sum_{i=1}^{m} l_i \rho_i a_i \tag{1.52a}$$

$$\text{s.c. :} \quad \sum_{i=1}^{m} r_i f_i = p, \tag{1.52b}$$

$$f_i = a_i (E_i/l_i) r_i^T u \qquad \forall i = 1, ..., m, \tag{1.52c}$$

$$a_i \geq 0 \qquad \forall i = 1, ..., m, \tag{1.52d}$$

$$a_i(\sigma_i^{min} - (E_i/l_i) r_i^T u) \leq 0 \qquad \forall i = 1, ..., m, \tag{1.52e}$$

$$a_i((E_i/l_i) r_i^T u - \sigma_i^{max}) \leq 0 \qquad \forall i = 1, ..., m, \tag{1.52f}$$

$$a_i(\sigma_i^{Cr}(a_i) - E_i/l_i) r_i^T u) \leq 0 \qquad \forall i = 1, ..., m, \tag{1.52g}$$

où les notations sont les mêmes que précédemment. On notera que les contraintes (1.52e), (1.52f) et (1.52g) sont les contraintes évanescentes associées à la contrainte $a_i \geq 0$.

1.3.1.3 Les exemples étudiés

Nous présentons dans ce qui suit les problèmes exacts sur lesquels nous avons effectué les tests numériques présentés dans le chapitre 3. Ce sont des problèmes de

minimisation de volume de structure, modélisés sous la forme (1.51) ou (1.52). Les seules différences entre les problèmes concernent donc le nombre de barres et par conséquent de variables. Nous introduisons ces problèmes dans l'ordre croissant du nombre de variables qu'ils mettent en jeu.

Le problème à 5 barres :
Une représentation schématique du problème est donnée à la figure 1.13. Deux des

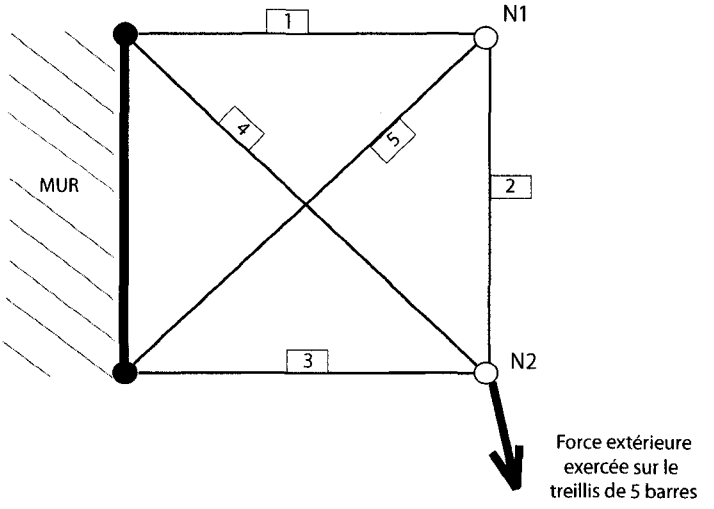


Figure 1.13 – Problème du volume minimum sur un treillis à 5 barres

noeuds du treillis sont ancrés dans un mur fixe. Les deux autres, N1 et N2, peuvent éventuellement se déplacer. Il y a donc quatre degrés de liberté :

- 2 pour le noeud $N1$ (un degré horizontal et un vertical)
- 2 pour le noeud $N2$ (un degré horizontal et un vertical).

Le vecteur u a 4 composantes. Le vecteur f, lui, a autant de composantes qu'il y a de barres. On se retrouve finalement avec 14 variables (5 pour a, 5 pour f et 4 pour u)

pour ce problème. Dans notre exemple, les valeurs des paramètres sont les suivantes :

$$\begin{aligned} l_1 = l_2 = l_3 &= 1, \\ l_4 = l_5 &= \sqrt{2}, \\ \sigma_i^{max} = -\sigma_i^{min} &= 5, \\ E_i &= 1, \\ P &= (0, 0, 5, -50), \end{aligned}$$

et on part de la situation initiale décrite à la figure 1.14.

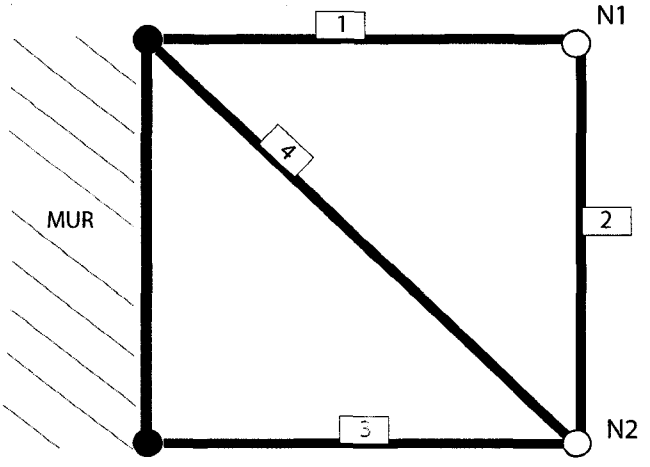


Figure 1.14 – Treillis initial à 5 barres (diamètres = 50)

Le problème à 10 barres :
Un schéma représentatif de ce problème est donné à la figure 1.15.

Dans ce cas, on a 8 degrés de liberté (4 noeuds et 2 degrés de liberté pour chacun d'eux) et deux charges appliquées. On a donc 28 variables pour cet exemple. Ce problème est tiré de l'article (Guo et al., 2001). On prend donc les mêmes données (longueur et masse volumique des barres, module d'Young, etc...) que dans l'article afin de pouvoir comparer les résultats.

Le problème à 18 barres :
On peut représenter schématiquement le problème par la figure 1.16.

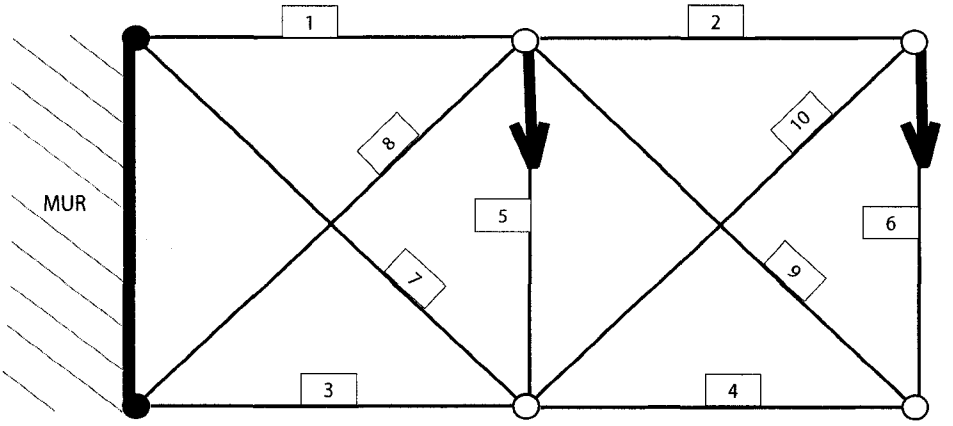


Figure 1.15 – Problème du volume minimum sur un treillis à 10 barres

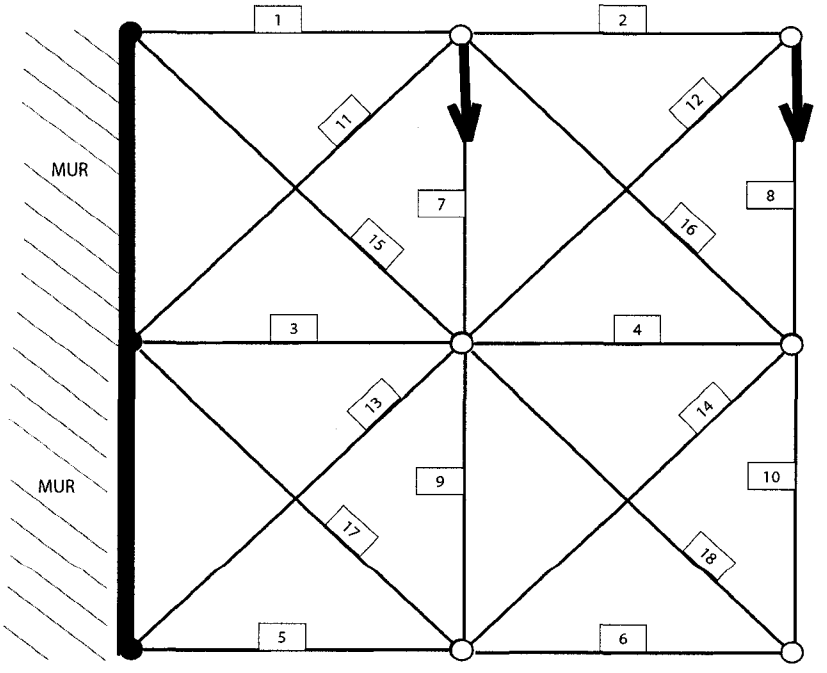


Figure 1.16 – Problème du volume minimum sur un treillis à 18 barres

Ce problème, que nous avons conçu par extension du précédent, possède lui 12 degrés de liberté (même raisonnement que précédemment). Cela nous amène à considérer 48 variables. On reprend les mêmes valeurs des paramètres que pour l'exemple à 10 barres. Le treillis initial considéré est donné à la figure 1.17. C'est donc

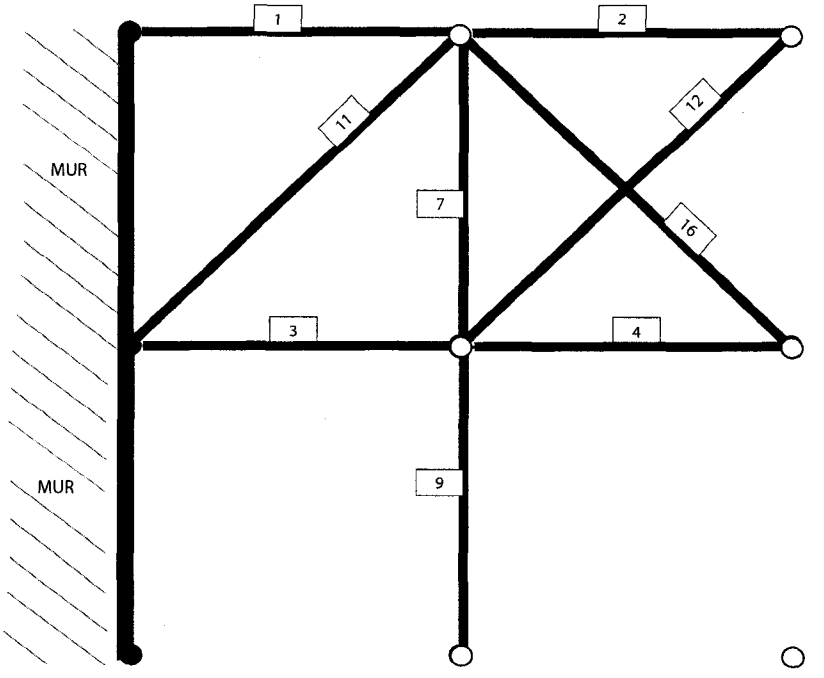


Figure 1.17 – Treillis initial à 18 barres (diamètres = 50)

sur ces problèmes précis que nous avons testé PSUPERB lors des tests numériques. Nous verrons au chapitre 3 qu'en dépit de leur simplicité apparente, ces exemples sont déjà difficiles à résoudre.

1.3.2 L'approche de l'énergie de déformation minimale

Cette approche consiste à minimiser l'énergie de déformation, C, de la structure. Le problème se pose alors dans les termes suivants :

$$\begin{aligned} \underset{a\in\mathbb{R}^m, u\in\mathbb{R}^n}{\text{minimiser}} \quad & C = p^T u \\ \text{s.c. :} \quad & (\sum_{i=1}^{m} a_i A_i)u = p, \\ & \sum_{i=1}^{m} a_i \leq 1, \\ & a_i \geq 0 \quad \forall i = 1, ..., m, \end{aligned} \tag{1.53}$$

où $A_i \in \mathbb{R}^{n\times m}$ est telle que $A_i = (E_i/l_i) r_i r_i^T$ avec les notations prédédemment employées (on a aussi $\sum_{i=1}^{m} A_i = A$ avec A la matrice de rigidité) et le reste des notations est le même que précédemment.

Le type de problèmes traités avec cette formulation peut se représenter comme à la sous-section 1.3.1.3. Dans (Kocvara & Outrata, 2006), les auteurs travaillent avec des treillis de tailles YxZ où Y donne le nombres de noeuds horizontaux et Z le nombre de noeuds verticaux. Les problèmes possèdent diverses tailles allant de l'académique 3x3 au plus réaliste 41x9.

D'expérience, ce problème formulé tel qu'en (1.53) est très difficile à résoudre d'un point de vue numérique. Pour preuve, on présente dans la table 1.1 les résultats numériques obtenus par (Kocvara & Outrata, 2006) avec des solveurs commerciaux (SNOPT : (Gill, Murray, & Saunders, 2005), KNITRO : (Byrd, Gilbert, & Nocedal, 2000b) et (Byrd, Hribar, & Nocedal, 1999b), MINOS : (Murtagh & Saunders, 1998)) qu'on peut essayer sur

`neos.mcs.anl.gov/neos/solvers/index.html`.

Tableau 1.1: Nombre d'itérations effectuées par les solveurs commerciaux sur des problèmes de type (1.53)

	SNOPT	KNITRO	MINOS
Treillis 3x3	70	échec	149
Treillis 4x4	299	échec	échec
Treillis 5x5	434	échec	échec
Treillis 6x2	722	échec	échec
Treillis 41x9	échec	échec	échec

On voit alors que les résultats obtenus sont mauvais et qu'à partir du moment où on s'intéresse à des treillis de taille réelle, aucun solveur ne parvient à trouver de solution optimale. (Kocvara & Outrata, 2006) essaient alors de formuler le problème différemment. Ils envisagent le problème comme un programme mathématique bi-niveau. Avant de continuer, expliquons brièvement ce qu'est la programmation bi-niveau.

Soient deux vecteurs de variables x et y. Soit alors le problème

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimiser}} & f(x,y) \\ \text{s.c. :} & h(x,y)=0, \\ & g(x,y)\geq 0, \end{array} \tag{1.54}$$

où $f:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}$, $h:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}^p$ et $g:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}^q$.
Notons alors $\Psi(y)$ l'ensemble des solutions de ce problème pour y fixé. Une solution particulière est notée $x(y)$. Le problème (1.54) est appelé le problème *esclave* ou *suiveur*. On considère ensuite le problème

$$\begin{array}{ll} \underset{y\in\mathbb{R}^m}{\text{minimiser}} & F(x(y),y) \\ \text{s.c. :} & H(x(y),y)=0, \\ & G(x(y),y)\geq 0, \end{array} \tag{1.55}$$

où $F:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}$, $h:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}^k$, $g:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}^l$ et $x(y)\in\Psi(y)$.
Le problème (1.55) est le problème de programmation bi-niveau proprement dit (encore appelé problème *maître* ou *meneur*). On comprend maintenant l'appelation bi-

niveau : la solution y^*, optimale, du problème bi-niveau (1.55) est un vecteur de paramètres pour le problème esclave (1.54), qui, conjointement avec $x(y^*) \in \Psi(y^*)$ permet de minimiser $F(x(y), y)$. *Pour résoudre le problème maître, il faut d'abord considérer un problème esclave.*

Il existe plusieurs moyens pour passer d'un problème bi-niveau à un programme mathématique ordinaire. L'un d'entre eux est de remplacer le problème esclave par ses conditions d'optimalité (conditions de KKT). C'est l'option prise par (Kocvara & Outrata, 2006). Le résultat est une formulation de type (MPEC) (voir section 1.1.3). Le problème de structure peut alors s'écrire sous la forme

$$\begin{aligned} \underset{a\in\mathbb{R}^m, u\in\mathbb{R}^n, \alpha\in\mathbb{R}}{\text{minimiser}} \quad & \mathcal{G}(a,u) \\ \text{s.c. :} \quad & (\sum_{i=1}^{m} a_i A_i)u = p, \\ & \sum_{i=1}^{m} a_i \leq 1, \\ & (\alpha - \tfrac{1}{2}u^T A_i u) \geq 0 \quad \forall i = 1, ..., m, \\ & a_i \geq 0 \quad \forall i = 1, ..., m, \\ & (\alpha - \tfrac{1}{2}u^T A_i u)a_i = 0 \quad \forall i = 1, ..., m, \end{aligned} \tag{1.56}$$

où α est une variable duale qui n'a pas d'interprétation physique et $\mathcal{G}$ est une fonction strictement convexe et continûement différentiable. Notons que cette formulation ne semble pas plus simple à résoudre que la précédente. Les tests numériques vont d'ailleurs dans ce sens. En revanche, l'intérêt de (1.56) est de servir de point de départ pour d'autres formulations plus simples, dans lesquelles par exemple on fixe a et on minimise sur u. Ces autres formulations ont alors des solutions numériques plus faciles à trouver et en résolvant plusieurs problèmes de ce type, on arrive trouver des solutions satisfaisantes, du point de vue pratique, au problème de départ.

En ce qui nous concerne, nous avons travaillé sur cette formulation MPEC du problème afin de tester les performances de notre algorithme sur des problèmes concrets. C'est pourquoi par la suite, nous avons effectués des essais sur une collection de problèmes de ce genre, en guise de préalable au traitement des problèmes de structure.

Chapitre 2

Implémentation de la méthode de pénalisation élastique

Afin d'implémenter l'algorithme présenté au chapitre 1, section 1.2.5, dans le but d'effectuer des tests numériques, nous avons utilisé plusieurs langages et environnements. Dans ce qui suit, nous allons préciser nos choix et décrire comment nous avons procédé pour coder la version actuelle de PSUPERB.

2.1 Implémentation de PSUPERB

2.1.1 Pourquoi Python ?

Pour développer le code nous avons choisi d'utiliser le langage de programmation Python. C'est un langage de programmation orienté objet dont les principaux avantages sont :

- la clarté de la syntaxe ;
- la facilité d'intégration de nouveaux modules ou d'interfaçage avec d'autres langages ;
- les nombreuses extensions numériques existantes ;
- la simplicité de la maintenance du code.

Nous avons retenu ce langage de programmation car il était parfaitement adapté à la situation. En effet, nous savions que nous allions faire appel à un langage de modélisation et que notre code devrait être capable d'interpréter les informations générées par ce langage de modélisation. Ensuite, le projet étant susceptible d'être développé par différents étudiants, il fallait trouver un langage qui permette à chacun de modifier sa partie tout en bénéficiant du travail de ses camarades. Enfin, une syn-

taxe claire et simple permet évidemment de maximiser l'efficacité du programmeur en lui permettant de se concentrer sur des problèmes dignes d'intérêt.

Donnons quelques exemples permettant d'illustrer nos propos. Le listing 2.1 est un extrait du code de PSUPERB.

Listing 2.1 – Extrait du code psuperbV1.py

```
#
# Main driver for Psuperb
#

import sys
import psuperbFrameworkV1 as S
import trustregion as T
import norms as N
import pygltr
import umfpack
```

```
def SolveInnerIteration( nlp, TR, x, s, t, y, yrange2, z, zrange2, u, v ):

    n_inner = 0
    finished = False
    tinyStep = False
    n_it_infeasible_inner = 0

    while not finished:

        res = nlp.InnerResiduals( x, s, t, y, yrange2, z, zrange2, u, v )

        if nlp.SufficientlySmall( res ):
            print'   STATUS'
            print'   Inner iteration stopping conditions satisfied'
            finished = True
        elif tinyStep:
            print
            print'   STATUS'
            print'   Exiting inner iteration---tiny step'
            finished = True

        else:
            (x, s, t, y, yrange2, z, zrange2, u, v, tinyStep, n_it_infeasible_spb)=\
                SolveSubproblem( nlp, TR, x, s, t, y, yrange2, z, zrange2, u, v )
            n_inner += 1
            n_it_infeasible_inner += n_it_infeasible_spb

            if n_inner >= nlp.max_inner:
                print ' Skipping to next outer iteration'
                finished = True
```

```
    # End while

    TR.ResetRadius()
    return(x,s,t,y,yrange2,z,zrange2,u,v,n_inner,n_it_infeasible_inner,tinyStep)
```

Essayons de comprendre rapidement ce qui est écrit. Les premières lignes

Listing 2.2 – Importation des modules

```
#
# Main driver for Psuperb
#

import sys
import psuperbFrameworkV1 as S
import trustregion as T
import norms as N
import pygltr
import umfpack
```

ne sont rien d'autres qu'une façon d'importer des modules grâce au mot-clé `import`. Par exemple, le module `umfpack` permet d'effectuer des factorisations de matrices creuses. Si on crée un nouveau module, il suffit donc d'utiliser le mot-clé adéquat et on peut immédiatement en exploiter tous les atouts. Dans notre cas, nous avons créé le module `precond` au bout de quelques mois de travail et en quelques secondes nous avons pu voir les changements apportés par l'utilisation d'un préconditionneur. Cette simplicité d'implémentation a donc été une composante essentielle de l'avancée efficace du projet. Ensuite, sans entrer dans les détails, on a

Listing 2.3 – Déclaration d'une méthode

```
def SolveInnerIteration( nlp, TR, x, s, t, y, yrange2, z, zrange2, u, v ):
```

qui est le moyen, grâce au mot-clé `def`, de déclarer une méthode ou une fonction. Dans l'exemple on déclare la méthode pour résoudre les itérations internes. Entre parenthèses, on donne les paramètres et variables d'entrée. La ligne

Listing 2.4 – Sortie

```
    return(x,s,t,y,yrange2,z,zrange2,u,v,n_inner,n_it_infeasible_inner,tinyStep)
```

permet de savoir quels sont les paramètres et variables renvoyés par la méthode `SolveInnerIteration`. Enfin, entre ces deux lignes de code réside la méthode proprement dite. On voit qu'il n'y a rien de nouveau par rapport à d'autres langages de programmation (JAVA par exemple) et qu'on utilise les classiques boucles `while` et

`for` ainsi que le test `if`. Il est à noter que l'élément qui permet de marquer le début et la fin des boucles est l'**indentation**. Nous venons ainsi de voir que n'importe quel utilisateur, familier avec la programmation informatique, est capable de prendre rapidement possession du code. Python étant aussi un langage orienté objet, il se prête parfaitement à l'implémentation d'un algorithme par assemblage de blocs de base. Toutes ces qualités réunies font de Python un langage d'exception dans le cadre de la recherche universitaire et c'est pourquoi nous l'avons choisi.

2.1.2 NLPy

Sur la base de la qualité et des avantages présentés par Python, un paquet d'optimisation non-linéaire développé par (Orban & Friedlander, 2003) est disponible sur `nlpy.sourceforge.net`. NLPy est un ensemble d'outils utiles au développement d'algorithmes d'optimisation. Ces outils sont presque intégralement codés en Python et sont principalement l'implémentation de méthodes d'optimisation non-linéaire (par exemple *méthode de gradient conjugué*, *méthode de région de confiance*, ...) ou des algorithmes complets pour résoudre des problèmes d'optimisation (par exemple, `trip.py` est un algorithme qui implémente une méthode de points intérieurs). Grâce à la nature de Python, on peut faire jouer ces modules entre eux en fonction des besoins de développement. Pour les algorithmes complets, les modèles doivent être écrits en AMPL et le fichier d'entrée doit être un `.nl` (voir la section 2.1.3 ou (Fourer, Gay, & Kernighan, 2003)). Ce fichier est alors interprété par `amplpy.py`. Notons que des routines sont implémentées pour l'utilisation de matrices creuses ce qui permet de traiter des problèmes de grande taille. Pour coder PSUPERB, on a utilisé les modules suivants :

- `amplpy.py`
- `trustregion.py`
- `trpcg.py`
- `precond.py`
- `psuperbFramework.py`
- `psuperb.py`

Comme nous l'avons dit, le premier permet d'interpréter le modèle et les données et les traduire en un format utilisable par les fichiers écrits en Python. Le deuxième et le

troisième implémentent respectivement une méthode de région de confiance et de gradient conjugué tronqué. Le quatrième est un module pour utiliser un préconditionneur (voir section 2.2.2). Le cinquième est un fichier qui constitue une sorte de canevas pour `psuperb.py`. En effet, on y inclut toutes les méthodes qui sont nécessaires à la résolution du problème par PSUPERB (calcul des valeurs des contraintes, calcul de la matrice Hessienne, ...). Ainsi, dans `psuperb.py` on ne garde que l'implémentation de l'algorithme proprement dit. On peut résumer tout ce qui précède graphiquement par la figure 2.1.

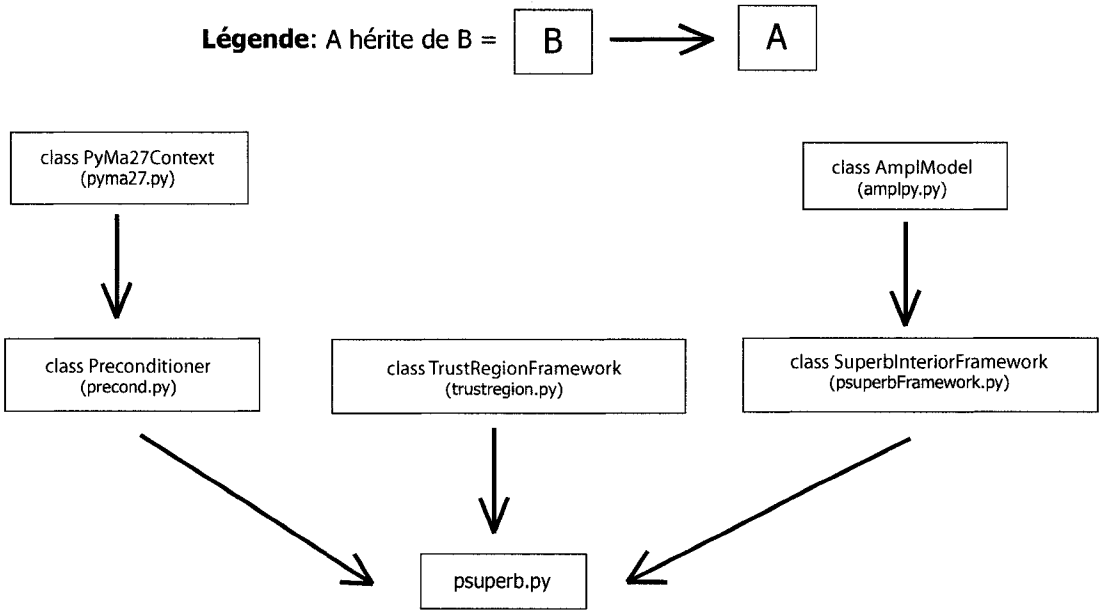


Figure 2.1 – Graphe de l'héritage dans NLPy pour l'implémentation de PSUPERB

C'est donc dans l'environnement NLPy que nous avons développé l'implémentation de PSUPERB. Nous allons à présent aborder le langage de modélisation dont nous nous sommes servi.

2.1.3 AMPL

Comme toujours en recherche opérationnelle nous avons eu besoin d'un langage de modélisation. Nous avons choisi AMPL (Fourer et al., 2003). Celui-ci s'est présenté naturellement puisqu'il est à la fois très simple à prendre en main, capable de traiter des problèmes de grande taille et qu'il était déjà utilisé par NLPy. Nous avons eu recours à AMPL pour modéliser les problèmes que nous voulions résoudre avec notre algorithme.

Dans ce langage, on distingue trois types de fichiers :

- le fichier de modèle, monFichierModele.mod
- le fichier de données, monFichierData.dat
- Un fichier nl, monFichierFinal.nl.

Le dernier fichier est créé à partir des deux autres en utilisant la commande *unix* suivante :

```
$ampl  -ogmonFichierFinal monFichierModele.mod monFichierData.dat
```

Dans le fichier de modèle, on écrit tout ce qui est intrinsèque au problème considéré, par exemple l'objectif ou les contraintes. Dans le fichier de données, on écrit tout ce qui est susceptible de changer en fonction du choix de l'utilisateur, typiquement, les paramètres. Le fichier nl permet d'assembler ce qui est contenu dans les deux autres et sera interprété par les solveurs.

Nous présentons un exemple de ce qui a pu être écrit pour coder le problème (1.51). Le modèle est donné au listing 2.5.

Listing 2.5 – trussVolume.mod : le modèle du problème (1.51) en AMPL

```
###Truss structure model

###Sets
set nb_bars;
set nb_degLibs;

###Parameters
param l{nb_bars};
param rho{nb_bars};
param E{nb_bars};
param sigma_min{nb_bars};
param sigma_max{nb_bars};
```

```
param p{nb_degLibs};
param vect_r{nb_bars, nb_degLibs};


###Vars
var a {nb_bars} >=0;
var f {nb_bars} ;
var u {nb_degLibs};


####Objective function
minimize obj:
         sum{i in nb_bars} a[i]*l[i]*rho[i];


###Constraints

subject to egalite_force {j in nb_degLibs}:
        (sum{i in nb_bars} vect_r[i,j]*f[i] ) = p[j];

subject to egalites_Hooke{i in nb_bars}:
         f[i] = a[i]*(E[i]/l[i])*( sum{j in nb_degLibs} vect_r[i,j]*u[j] );

subject to minor_f {i in nb_bars}:
         a[i]*sigma_min[i] <= f[i];

subject to major_f {i in nb_bars}:
        0 >= f[i] - a[i]*sigma_max[i];

subject to bornes_inf_f {i in nb_bars}:
        0 <= (a[i]^2)*3.14159265*E[i]/(4*l[i]^2) + f[i];
```

Comme on peut le voir, hormis certains mots-clé comme par exemple `var` (pour déclarer les variables), `minimize` (pour savoir si on maximise ou minimise) ou `subject to` (pour déclarer les contraintes) tout le reste du code est parfaitement transparent. Il est donc très simple d'interpréter ce qui est écrit et donc de travailler avec des modèles écrits par d'autres personnes. Les données sont présentées au listing 2.6.

Listing 2.6 – truss5Bars.dat : les données du problème (1.51) en AMPL pour un treillis à 5 barres

```
data;

set nb_bars :=  BAR_1 BAR_2  BAR_3  BAR_4  BAR_5;
set nb_degLibs := degLib_1 degLib_2 degLib_3 degLib_4 ;

param:  E       rho     sigma_min       sigma_max       l       f               a :=
BAR_1   1       1       -5              5               1       50              50
BAR_2   1       1       -5              5               1       50              50
```

```
BAR_3   1       1       -5              5               1       5               50
BAR_4   1       1       -5              5               1.414   -70.7107        50
BAR_5   1       1       -5              5               1.414   0               0;

param: p u:=
        degLib_1        0               1
        degLib_2        0               -3.828
        degLib_3        5               0.1
        degLib_4        -50             -4.828
        ;

param vect_r[*,*] :=

                # BARRE 1                       # BARRE 2
                BAR_1 degLib_1 1                BAR_2 degLib_1 0
                BAR_1 degLib_2 0                BAR_2 degLib_2 1
                BAR_1 degLib_3 0                BAR_2 degLib_3 0
                BAR_1 degLib_4 0                BAR_2 degLib_4 -1

                # BARRE 3                       # BARRE 4
                BAR_3 degLib_1 0                BAR_4 degLib_1 0.70710678118655
                BAR_3 degLib_2 0                BAR_4 degLib_2 0.70710678118655
                BAR_3 degLib_3 1                BAR_4 degLib_3 0
                BAR_3 degLib_4 0                BAR_4 degLib_4 0

                # BARRE 5
                BAR_5 degLib_1 0
                BAR_5 degLib_2 0
                BAR_5 degLib_3 0.70710678118655
                BAR_5 degLib_4 -0.70710678118655;
```

Pour les données, il y a aussi quelques mots-clé à connaître pour comprendre de quoi il s'agit. Par exemple, le mot `set` sert à définir les ensembles, le mot `param` à définir un paramètre.

Les fichiers de modèle et données présentés ici sont très simples mais on peut intégrer toutes sortes de complications sans le moindre problème comme on peut le constater à la lecture de (Fourer et al., 2003) ou à la consultation de `www.ampl.com`.

2.1.4 Exemple complet : le développement de PSUPERB

Maintenant qu'on a présenté les différents langages, nous allons nous pencher sur l'exemple du développement de l'algorithme PSUPERB qui a nécessité l'utilisation de chacun d'eux. Cet algorithme vise à résoudre le problème (NLP). Nous proposons dans ce qui suit un tour d'horizon des fichiers essentiels ainsi que leur rapport avec le problème de départ et/ou l'algorithme. Enfin nous donnerons la procédure à suivre

pour utiliser le code sur l'exemple d'un problème de structure à 5 barres.

2.1.4.1 amplpy.py

Dans cette partie , nous présentons les notations ainsi que tout le formalisme qui a été utilisé lors de l'implémentation. Dans le code, on perçoit le problème (NLP) de la manière suivante

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimiser}} & f(x) \\ \text{s.c. :} & c_{\mathcal{E}}(x) = \gamma^E, \\ & \gamma^L \leq c_{\mathcal{I}}(x) \leq \gamma^U, \\ & x^L \leq x \leq x^U, \end{array} \tag{2.1}$$

où $f : \mathbb{R}^n \to \mathbb{R}$, $c_{\mathcal{E}} : \mathbb{R}^n \to \mathbb{R}^{n_{\mathcal{E}}}$, $c_{\mathcal{I}} : \mathbb{R}^n \to \mathbb{R}^{n_{\mathcal{I}}}$ sont des fonctions de classe $\mathcal{C}^2$ et $\gamma^E \in \mathbb{R}^{n_{\mathcal{E}}}$, $\gamma^U, \gamma^L \in \mathbb{R}^{n_{\mathcal{I}}}$ et $x^U, x^L \in \mathbb{R}^n$ sont des vecteurs fixés. E et I sont des ensembles finis d'indices.

On définit alors les ensembles suivants

$$\mathcal{C}^L = \{i \in \mathcal{I} \mid -\infty < \gamma_i^L \text{ et } \gamma_i^U = +\infty\}, \tag{2.2a}$$
$$\mathcal{C}^U = \{i \in \mathcal{I} \mid -\infty = \gamma_i^L \text{ et } \gamma_i^U < +\infty\}, \tag{2.2b}$$
$$\mathcal{C}^R = \{i \in \mathcal{I} \mid -\infty < \gamma_i^L \text{ et } \gamma_i^U < +\infty\}, \tag{2.2c}$$

qu'on appelle respectivement l'ensemble des **contraintes inférieures**, l'ensemble des **contraintes supérieures** et l'ensemble des **doubles contraintes**.
De la même manière on définit

$$\mathcal{B}^L = \{i \in \mathcal{I} \mid -\infty < x_i^L \text{ et } x_i^U = +\infty\}, \tag{2.3a}$$
$$\mathcal{B}^U = \{i \in \mathcal{I} \mid -\infty = x_i^L \text{ et } x_i^U < +\infty\}, \tag{2.3b}$$
$$\mathcal{B}^R = \{i \in \mathcal{I} \mid -\infty < x_i^L \text{ et } x_i^U < +\infty\}, \tag{2.3c}$$

qu'on appelle respectivement l'ensemble des **contraintes de bornes inférieures**, l'ensemble des **contraintes de bornes supérieures** et l'ensemble des **doubles contraintes de bornes**.

On pose alors $\mathcal{C} = \mathcal{C}^L \cup \mathcal{C}^U \cup \mathcal{C}^R \cup \mathcal{E}$, $\mathcal{B} = \mathcal{B}^L \cup \mathcal{B}^U \cup \mathcal{B}^R$, $n_{\mathcal{C}} = |\mathcal{C}|$ et $n_{\mathcal{B}} = |\mathcal{B}|$.

Toutes ces informations sont contenues dans le fichier *.nl*. Le rôle de `amplpy.py` est donc, en partie, de lire ces informations et de les rendre utilisables par les autres modules. On crée donc une classe, `AmplModel`, qui récupère les informations et les traduit en langage Python. Ce qui a été écrit plus haut se traduit en termes de membres et de méthodes de la classe `AmplModel` :

- La fonction objectif est appelée `obj`
- La valeur initiale du vecteur de variables x est contenu dans x_0
- Les bornes inférieures et supérieures pour les contraintes sont rangées dans `Lvar, Uvar, Lcon` et `Ucon` respectivement pour x^L, x^U, γ^L et γ^U
- On appelle `lowerC, upperC, rangeC` et `equalC`, respectivement les ensembles $\mathcal{C}^L$, $\mathcal{C}^U$, $\mathcal{C}^R$ et $\mathcal{E}$
- On appelle `lowerB, upperB` et `rangeB`, respectivement les ensembles $\mathcal{B}^L$, $\mathcal{B}^U$ et $\mathcal{B}^R$.

On récupère aussi d'autres précieuses informations fournies par AMPL comme par exemple, le gradient et le Hessien de l'objectif, la valeur des contraintes en un point donné, etc... Ce qu'il faut bien comprendre c'est que tout module qui hérite de la classe `AmplModel` (voir figure 2.1) a accès à ces informations. C'est le cas de `psuperbFramework.py`.

2.1.4.2 psuperbFramework.py

Nous allons maintenant expliquer le contenu du fichier `psuperbFramework.py`. D'abord, on crée une classe `SuperbInteriorFramework` qui hérite de `AmplModel`. On a donc accès à toutes les données présentées ci-dessus. Dans cette classe on définit tout ce qui est nécessaire à la résolution du problème : les variables primales, duales, la fonction barrière, son gradient, son Hessien, ...

Lorsqu'on applique PSUPERB à (2.1), on doit introduire des variables élastiques (voir l'explication dans la section 1.2.4, Méthode élastique). Dans l'implémentation, on a choisi de distinguer les contraintes de bornes des autres contraintes car AMPL effectue déjà ce travail. On donc a trois types de variables primales :

- les variables du problème initial qu'on note `x`
- les variables élastiques associées aux contraintes de bornes sur x, appelées `t`
- les variables élastiques associées aux autres contraintes sur x, on les appelle `s`.

On introduit alors les variables duales suivantes :

- les variables duales associées aux contraintes générales (ensemble $\mathcal{C}$) sur x, qu'on nomme `y`
- les variables duales associées aux contraintes de bornes (ensemble $\mathcal{B}$) sur x, qu'on nomme `z`
- les variables duales associées aux contraintes sur s, appelées `u`
- les variables duales associées aux contraintes sur t, appelées `v`.

En fait, on a deux autres variables duales qu'on utilise lorsqu'il y a des doubles contraintes de bornes ou doubles containtes générales :

- les variables duales associées aux contraintes de borne supérieure des doubles contraintes générales sur `x`, qu'on nomme `yrange2`
- les variables duales associées aux contraintes de borne supérieure des doubles contraintes de borne sur `x`, qu'on nomme `zrange2`.

Notons que lorsqu'il y a des doubles contraintes, `y` et `z` représentent les variables duales pour les contraintes de la borne inférieure. Remarquons aussi que dans le cas des doubles contraintes, on n'a besoin que d'une seule variable élastique.

Exemple

Soit le problème

$$\begin{aligned} \underset{x_1,x_2}{\text{minimiser}} \quad & (x_1-1)^2 \\ \text{s.c. :} \quad & -2 \leq x_2^2 + x_1 \leq 8, \quad \text{(c1)} \\ & 3 \leq x_2 \leq 18. \quad \text{(c2)} \end{aligned}$$

Pour ce problème, on définit donc le jeu de variables suivant :

- un vecteur $x \in \mathbb{R}^2$ qui correspond à (x_1, x_2)
- une variable s_1 associé à la contrainte c1
- une variable t_1 associée à la contrainte c2
- une variable duale y pour $-2 \leq x_2^2 + x_1$
- une variable duale $yrange2$ pour $x_2^2 + x_1 \leq 8$

- une variable duale z pour $3 \leq x_2$
- une variable duale $zrange2$ pour $x_2 \leq 18$
- une variable duale u_1 pour la contrainte associée à s_1 ($s_1 \geq 0$, voir la formulation (1.26))
- une variable duale v_1 pour la contrainte associée à t_1 ($t_1 \geq 0$).

Par ailleurs, les contraintes sont définies comme suit

$$con_i(x) = \begin{cases} c_i(x) - \gamma_i^E & i \in equalC, \\ c_i(x) - \gamma_i^L & i \in lowerC, \\ \gamma_i^U - c_i(x) & i \in upperC, \\ c_i(x) - \gamma_i^L & i \in rangeC, \end{cases} \quad \text{et} \quad conRange_i(x) = \gamma_i^U - c_i(x)\ i \in rangeC.$$

On définit ensuite les deux paramètres fondamentaux de l'algorithme :

- μ, le paramètre pour les sous-problèmes barrière ((voir la section sur les rappels sur les méthodes de points intérieurs, équation (1.15))
- ν, le paramètre de pénalité. On a d'ailleurs adopté un ν différent selon les contraintes. Ainsi, en se référant à la formulation (1.26), on a un

  ν_{Eq} qui pénalise les $c_i(x) + 2s_i,\ i \in equalC$

  ν_S qui pénalise les $s_i,\ i \in lowerC \cup upperC \cup rangeC$

  ν_T qui est l'équivalent du ν_S pour les contraintes de bornes (par conséquent associé aux variables `t`).

  On a donc $\nu = (\nu_{Eq}, \nu_S, \nu_T)$.

À présent, on peut explicitement donner la manière de calculer les estimateurs des multiplicateurs de Lagrange primaux pour les contraintes générales :

$$\begin{aligned}
y_i^E &= \mu(c_i(x) - \gamma_i^E + s_i)^{-1} && i \in \mathcal{E} \text{ (c'est-à-dire } i \in equalC), \\
y_i^L &= \mu(c_i(x) - \gamma_i^L + s_i)^{-1} && i \in \mathcal{C}^L \text{ (c'est-à-dire } i \in lowerC), \\
y_i^U &= \mu(\gamma_i^U - c_i(x) + s_i)^{-1} && i \in \mathcal{C}^U \text{ (c'est-à-dire } i \in upperC), \\
y_i^{RL} &= \mu(c_i(x) - \gamma_i^L + s_i)^{-1} && i \in \mathcal{C}^R \text{ (c'est-à-dire } i \in rangeC), \\
y_i^{RU} &= \mu(\gamma_i^U - c_i(x) + s_i)^{-1} && i \in \mathcal{C}^R \text{ (c'est-à-dire } i \in rangeC),
\end{aligned}$$

pour les contraintes de bornes :

$$\begin{aligned}
z_i^L &= \mu(x_i - x_i^L + t_i)^{-1} \quad i \in \mathcal{B}^L \text{ (c'est-à-dire } i \in lowerB), \\
z_i^U &= \mu(x_i^U - x_i + t_i)^{-1} \quad i \in \mathcal{B}^U \text{ (c'est-à-dire } i \in upperB), \\
z_i^{RL} &= \mu(x_i - x_i^L + t_i)^{-1} \quad i \in \mathcal{B}^R \text{ (c'est-à-dire } i \in rangeB), \\
z_i^{RU} &= \mu(x_i^U - x_i + t_i)^{-1} \quad i \in \mathcal{B}^R \text{ (c'est-à-dire } i \in rangeB),
\end{aligned}$$

pour les contraintes sur les variables élastiques `s` associées aux contraintes générales :

$$u = S^{-1}\mu e$$

où S est une matrice diagonale (voir notation 1.2.2.1) et pour les contraintes sur les variables élastiques `t` associées aux contraintes de bornes :

$$v = T^{-1}\mu e.$$

Par la suite, les estimateurs primaux sont notés $\tilde{\mathtt{y}}$, $\tilde{\mathtt{z}}$, $\tilde{\mathtt{u}}$, $\tilde{\mathtt{v}}$. Les multiplicateurs décalés (voir notation 1.2.5.3) sont donnés par

$$\lambda_i(y,\nu) = \begin{cases} y_i^E - \nu_{Eq} & i \in \mathcal{E}, \\ y_i^L & i \in \mathcal{C}^L, \\ -y_i^U & i \in \mathcal{C}^U, \\ y_i^{RL} - y_i^{RU} & i \in \mathcal{C}^R \end{cases} \quad \text{et } \xi_i(z) = \begin{cases} z_i^L & i \in \mathcal{B}^L, \\ z_i^U & i \in \mathcal{B}^U, \\ z_i^{RL} - z_i^{RU} & i \in \mathcal{B}^R, \\ 0 \text{ sinon.} \end{cases} \tag{2.4}$$

On notera encore $\theta \in \mathbb{R}^{nc}$ la grandeur définie comme suit

$$\theta_i(x,\tilde{y},s) = \begin{cases} \tilde{y}_i^E/(c_i(x) - \gamma_i^E + s_i) & i \in \mathcal{E}, \\ \tilde{y}_i^L/(c_i(x) - \gamma_i^L + s_i) & i \in \mathcal{C}^L, \\ \tilde{y}_i^U/(\gamma_i^U - c_i(x) + s_i) & i \in \mathcal{C}^U, \\ \tilde{y}_i^{RL}/(c_i(x) - \gamma_i^{RL} + s_i) + \tilde{y}_i^{RU}/(\gamma_i^{RU} - c_i(x) + s_i) & i \in \mathcal{C}^R, \end{cases} \tag{2.5}$$

à laquelle on associera la matrice $\Theta(x,\tilde{y},s)$ (voir notation 1.2.2.1) et la grandeur

$\hat{\theta} \in \mathbb{R}^{n_C}$ définit par

$$\hat{\theta}_i(x, \tilde{y}, s) = \begin{cases} \tilde{y}_i^E/(c_i(x) - \gamma_i^E + s_i) & i \in \mathcal{E}, \\ \tilde{y}_i^L/(c_i(x) - \gamma_i^L + s_i) & i \in \mathcal{C}^L, \\ -\tilde{y}_i^U/(\gamma_i^U - c_i(x) + s_i) & i \in \mathcal{C}^U, \\ \tilde{y}_i^{RL}/(c_i(x) - \gamma_i^{RL} + s_i) - \tilde{y}_i^{RU}/(\gamma_i^{RU} - c_i(x) + s_i) & i \in \mathcal{C}^R, \end{cases} \tag{2.6}$$

à laquelle on associera la matrice $\hat{\Theta}(x, \tilde{y}, s)$. De la même manière on parlera de $\xi \in \mathbb{R}^{n_B}$

$$\xi_i(x, \tilde{z}, t) = \begin{cases} \tilde{z}_i^L/(x_i - x^L + t_i) & i \in \mathcal{B}^L, \\ \tilde{z}_i^U/(x_i^U - x_i + t_i) & i \in \mathcal{B}^U, \\ \tilde{z}_i^{RL}/(x_i - x_i^{RL} + t_i) + \tilde{z}_i^{RU}/(x_i^{RU} - x_i + t_i) & i \in \mathcal{B}^R, \end{cases}$$

et de $\hat{\xi} \in \mathbb{R}^{n_B}$

$$\hat{\xi}_i(x, \tilde{z}, t) = \begin{cases} \tilde{z}_i^L/(x_i - x^L + t_i) & i \in \mathcal{B}^L, \\ -\tilde{z}_i^U/(x_i^U - x_i + t_i) & i \in \mathcal{B}^U, \\ \tilde{z}_i^{RL}/(x_i - x_i^{RL} + t_i) - \tilde{z}_i^{RU}/(x_i^{RU} - x_i + t_i) & i \in \mathcal{B}^R, \end{cases} \tag{2.7}$$

auxquelles seront associées respectivement les matrices $\Xi(x, \tilde{z}, t)$ et $\hat{\Xi}(x, \tilde{z}, t)$.

Notation 2.1.4.1 *(Matrice E)*
Soit la matrice E de $\mathbb{R}^{n_B \times n_C}$ telle que $E^T \Xi(x, \tilde{z}, t) E$ soit diagonale et dont le i-ème terme soit $\xi_i(x, \tilde{z}, t)$ si $i \in \mathcal{B}$ et 0 sinon.

Notation 2.1.4.2 *(ϕ^B)*
On note ϕ^B la fonction objectif du sous-problème barrière (voir la section sur les rappels sur les méthodes de points intérieurs, équation (1.15)) associé au problème initial (2.1).

De façon explicite, on a

$$\begin{aligned}
\phi^B(x,s,t;\mu,\nu) \;=\; f(x) \;+\; & \sum_{i\in\mathcal{E}} \nu_{Eq}(c_i(x) - \gamma_i^E + 2s_i) + \sum_{i\in\mathcal{C}} \nu_S s_i + \sum_{i\in\mathcal{B}} \nu_T t_i \\
& - \sum_{i\in\mathcal{E}} \mu \log(c_i(x) - \gamma_i^E + s_i) - \sum_{i\in\mathcal{E}} \mu \log(s_i) \\
& - \sum_{i\in\mathcal{C}^L\cup\mathcal{C}^R} \mu \log(c_i(x) - \gamma_i^L + s_i) - \sum_{i\in\mathcal{C}} \mu \log(s_i) \\
& - \sum_{i\in\mathcal{C}^U\cup\mathcal{C}^R} \mu \log(\gamma_i^U - c_i(x) + s_i) \\
& - \sum_{i\in\mathcal{B}^L\cup\mathcal{B}^R} \mu \log(x_i - x_i^L + t_i) - \sum_{i\in\mathcal{B}} \mu \log(t_i) \\
& - \sum_{i\in\mathcal{B}^U\cup\mathcal{B}^R} \mu \log(x_i^U - x_i + t_i).
\end{aligned} \tag{2.8}$$

On trouve ensuite dans le code des fonctions qui calculent $\nabla\phi^B$, le gradient de ϕ^B ainsi que H, son Hessien puis encore d'autres fonctions qui permettent par exemple de vérifier que les conditions d'arrêt de la boucle interne sont satisfaites, de calculer les normes des vecteurs, etc... Toutes ces fonctions seront ensuite appelées par `psuperb.py`.

2.1.4.3 psuperb.py

Après cette présentation de tout ce qui est nécessaire à l'algorithme proprement dit, on peut enfin évoquer ce dernier. Son implémentation est contenue dans `psuperb.py`. Regardons alors ce que contient ce fichier. D'abord, on instancie une classe de `SuperbInteriorFramework` afin de pouvoir utiliser tout ce qu'on a mentionné plus haut. Ensuite, on trouve principalement trois fonctions :

- la fonction associée à la boucle externe : `SolveOuterIteration`
- celle liée à la boucle interne : `SolveInnerIteration`
- celle en rapport avec la résolution du sous-problème de région de confiance : `SolveSubproblem`

Pour être un peu plus précis, voici comment fonctionne chacune d'elle.

`SolveOuterIteration` :
Tant que les conditions d'optimalité de KKT pour le problème (1.26) ne sont pas satisfaites, on rentre dans la boucle interne, c'est-à-dire qu'on appelle `SolveInnerIteration`.

`SolveInnerIteration` :
Tant que les conditions d'optimalité (1.39) pour le sous-problème barrière (1.29) ne sont pas satisfaites, on résout une nouvelle itération de sous-problème barrière en appelant `SolveSubproblem`.

`SolveSubproblem` :
Cette fonction résout le sous-problème barrière (voir la section 1.2.5 pour des rappels sur les mécanismes qui permettent de résoudre le sous-problème), c'est-à-dire qu'elle minimise la fonction barrière ϕ^B jusqu'à ce que l'itéré courant satisfasse les conditions (1.39). Elle cherche donc les solutions à une série de problèmes du type

$$\underset{d \in \mathcal{B}(\Delta)}{\text{minimiser}} \ \nabla_{v_P} \phi^B(x, s; \mu, \nu) d + \tfrac{1}{2} d^T H^{PD} d, \tag{2.9}$$

où :

$$v_P = (x, s, t),$$

$$\mathcal{B}(\Delta) = \{d \in \mathbb{R}^{n+n_C+n_B} \mid ||d||_M \leq \Delta\},$$

avec $||.||_M$ une norme de mise à l'échelle associée à la matice M (voir Section 2.2.2) et

$$H^{PD} = \begin{bmatrix} H(x, \lambda(y,\nu)) + J^T(x)\Theta(v)J(x) + E^T(x)\Xi(v)E(x) & J^T(x)\hat{\Theta}(v) & E^T(x)\hat{\Xi}(v) \\ \hat{\Theta}(v)J(x) & \Theta(v) + US^{-1} & 0 \\ \hat{\Xi}(v)E & 0 & \Xi(v) + VT^{-1} \end{bmatrix}.$$

Le problème (2.9) revient à en quelque sorte à résoudre le système de Newton

$$H^{PD} d = -\nabla_{v_P} \phi^B(x, s; \mu, \nu)$$

de manière inexacte. On est donc en face d'un problème du type « *Trouver X tel que $AX = b$*» et pour cela, on sait qu'on peut utiliser une méthode de Gradient Conjugué (méthode itérative très efficace pour résoudre ce genre de problèmes). Dans le code,

on appelle la routine `trpcg.py` qui est l'implémentation d'une méthode de Gradient Conjugué. On peut résumer l'implémentation de PSUPERB par l'algorithme 2.1.1.

Remarque : On peut préciser la manière exacte dont le paramètre μ est traité dans le code actuel de PSUPERB. Pour la valeur initiale, on prend $\mu_0 = 5$. Ensuite, pour la mise à jour, on utilise la règle suivante : $\mu_{k+1} = 0.2\mu_k$. Ce sont des valeurs qui semblent faire en sorte que l'algorithme se comporte bien dans de nombreuses situations. Cependant, comme nous l'expliquerons au chapitre 3 section 3.4.2.2, on pourrait travailler sur l'optimisation de ces valeurs.

2.1.4.4 Procédure

On cherche ici à montrer un exemple de la procédure complète à suivre pour résoudre un problème. On prendra l'exemple du problème de minimisation du volume d'un treillis à 5 barres avec une contrainte extérieure de charge. L'illustration de ce problème est donné par la figure (1.13).

Modéliser le problème - Il faut d'abord modéliser le problème grâce à AMPL. Les listings 2.5 (modèle) et 2.6 (données) donnent explicitement le code qui modélise le problème auquel on s'intéresse.

Créer le .nl - En utilisant la commande

```
ampl -ogpb5 trussVolume.mod truss5Bars.dat
```

on crée le fichier qui sera lu par le solveur.

Lancer PSUPERB - On lance un Terminal et le code s'exécute par la commande

```
python psuperb.py pb5.nl
```

Lire les résultats - Il ne reste alors plus qu'à lire les résultats sur le Terminal.

Opérer des changements - En fonction des résultats obtenus, on peut chercher à modifier des paramètres dans l'un des fichiers suivants :
`psuperb.py`, `psuperbFramework.py`, `amplpy.py`, `trustregion.py`. Une fois les modifications apportées, on relance le code. On peut aussi modifier les données dans `pb5.dat` et il faut alors recréer le fichier `.nl`.

Algorithme 2.1.1 PSUPERB Implémenté

Étape 0. (Initialisation)

Choisir des valeurs initiales x^0, s^0, t^0, ν^0 , μ^0 et Δ^0.
Choisir des constantes $\gamma_1 > 1 > \gamma_2 > 0$ et $\Delta_{max} > 0$.
Poser $k = 0$.

Étape 1. (SolveOuterIteration)

Si les conditions de KKT pour (NLP) ne sont pas satisfaites :
- incrémenter k ;
- mettre à jour ν ;
- mettre à jour μ ;
- aller à Étape 2.

Sinon : STOP.

Étape 2. (SolveInnerIteration)

Si les conditions (1.39) ne sont pas satisfaites :
- aller à Étape 3.

Sinon : aller à Étape 1.

Étape 3. (SolveSubproblem)

Résoudre une itération de sous-problème barrière grâce à TRPCG.
Obtenir un pas d.
Si d produit une décroissance suffisante de la fonction barrière :
-aller à Étape 4.
Sinon : aller à Étape 5.

Étape 4. (Mise à jour positive de la Région de Confiance)

Mettre à jour x^k, s^k, t^k.
On augmente le rayon de la région de confiance : $\Delta^k = min(\gamma_1 \Delta^k, \Delta_{max})$.
Aller à Étape 2.

Étape 5. (Mise à jour négative de la Région de Confiance)

On diminue le rayon de la région de confiance : $\Delta^k = \gamma_2 \Delta^k$.
Aller à Étape 2 .

Avec ces informations, le lecteur peut aisément résoudre ses propres problèmes d'optimisation à l'aide de notre code.

2.2 Stratégies d'amélioration

2.2.1 Traitement du paramètre de pénalisation

Dans l'algorithme, deux paramètres sont mis en évidence : μ associé à la méthode barrière et ν associé à la fonction de mérite. Dans les applications, il s'est vite avéré que la gestion du paramètre ν avait une influence cruciale sur le bon déroulement de l'algorithme.

Exemple
Considérons le problème HS006 issu de la collection (Hock & Schittkowski, 1981) :

$$\begin{array}{ll} \underset{(x_1,x_2)}{\text{minimiser}} & (1-x_1)^2 \\ \text{s.c. :} & 10(x_2-x_1^2)=0. \end{array} \tag{2.10}$$

Pour ce problème, si on prend un ν initial tel que $\nu = (\nu_{Eq}, \nu_S, \nu_T) = (100, 100, 100)$, l'algorithme échoue. En revanche, pour $\nu = (\nu_{Eq}, \nu_S, \nu_T) = (5, 1, 1)$, l'algorithme converge vers la solution (1,1) en une seconde environ.

Nous nous sommes donc intéressés aux valeurs qu'il fallait donner à ν pour que l'algorithme converge pour un nombre significatif de problèmes. Nous avons donc tenté de proposer des heuristiques pour choisir le ν initial ainsi qu'une règle pour mettre à jour le ν au cours des itérations. On propose dans ce qui suit deux heuristiques pour calculer la valeur du ν initial.

Heuristique n°1 : utiliser le gradient de l'objectif
La première méthode est issue de l'observation suivante. Quand on regarde la forme pénalisée (1.25) du problème (1.24), on voit qu'une condition nécessaire pour une solution de ce problème est de satisfaire

$$0 = \nabla f(x) + \nu Q(x),$$

où Q est une quantité qui dépend du point où on se trouve (on ne peut pas différencier en tout point à cause de la valeur absolue). Il apparaît donc un lien entre ν et $\nabla f(x)$. On détient alors la première idée pour proposer un paramètre ν initial. On posera

$$\nu_{Eq} = \nu_S = \nu_T = \gamma \max(1, ||\nabla f(x_0)||),$$

où γ est un paramètre strictement positif et x_0 l'itéré primal initial. L'utilisation de $\max(1, ||\nabla f(x0)||$ se justifie par le fait qu'on veut un paramètre ν strictement positif. Avec cette expression, on s'assure que cette propriété est vérifiée (même dans le cas où $||\nabla f(x_0)|| = 0$).

Exemple

Revenons maintenant à l'exemple du problème (2.10), en prenant comme point de départ $x_0 = (x_1, x_2) = (-1.2, 1)$. On a alors

$$\nabla f(x_0) = (4.4, 0),$$

d'où $\nu_{Eq} = \nu_S = \nu_T = ||(4.4, 0)|| = 4.4$ (en prenant la norme euclidienne).

Heuristique n°2 : utiliser les conditions de KKT

Dans cette méthode, on manipule les relations sur les multiplicateurs de Lagrange pour en tirer une expression de ν. On utilise la relation :

$$||\nu^k e - (y^{k+1} - \nu^k e_{\mathcal{E}}^0) - u^{k+1}|| = 0,$$

valable à l'optimalité. Cela nous dit que si on se trouve en un point optimal, on a

$$\begin{aligned}
2\nu_{Eq} &= y_i + u_i \quad \forall i \in \mathcal{E}, \\
\nu_S &= y_i + u_i \quad \forall i \in \mathcal{C}, \\
\nu_S &= y_i + yrange2_i + u_i \quad \forall i \in \mathcal{C}^R, \\
\nu_T &= z_i + v_i \quad \forall i \in \mathcal{B}, \\
\nu_T &= z_i + zrange2_i + v_i \quad \forall i \in \mathcal{B}^R,
\end{aligned}$$

avec le formalisme expliqué dans la section 2.1.4. Ces relations nous fournissent une expression pour ν qu'on utilise pour calculer le ν initial. Cependant, les $y_i + u_i$ ne

sont pas toujours égaux entre eux (surtout à la première itération !) et donc il faut faire un choix pour le nu initial. On prend donc :

$$\begin{aligned}
&\nu_{Eq} = \max(1, \tfrac{1}{2}||(y_{e\in\mathcal{E}} + u_{e\in\mathcal{E}})||_\infty),\\
&\nu_S = \max(1, ||y_{i\in\mathcal{C}} + u_{i\in\mathcal{C}}||_\infty, ||y_{i\in\mathcal{C}^R} + yrange2_{i\in\mathcal{C}^R} + u_{i\in\mathcal{C}^R}||_\infty),\\
&\nu_T = \max(1, ||z_{i\in\mathcal{B}} + v_{i\in\mathcal{B}}||_\infty, ||z_{i\in\mathcal{B}^R} + zrange2_{i\in\mathcal{B}^R} + v_{i\in\mathcal{B}^R}||_\infty),
\end{aligned}$$

Exemple
Considérons toujours le même exemple. Il n'y a qu'une seule contrainte et c'est une égalité donc les ensembles $\mathcal{C}$ et $\mathcal{B}$ sont vides, d'où :

$$\begin{aligned}
&\nu_{Eq} = \max(1, \tfrac{1}{2}||10 + 0.22...||_\infty) = 5.11...,\\
&\nu_S = 1,\\
&\nu_T = 1.
\end{aligned}$$

2.2.2 Préconditionnement

Ici, on s'intéresse au sous-problème de région de confiance. On a vu, lors du rappel sur les méthodes de région de confiance du chapitre 1, que cela consiste en partie à résoudre le problème (1.13), ce qui revient à se pencher sur la résolution du système $\nabla f(x)^T = -H_k s$. Pour cela, on sait qu'on dispose de diverses méthodes mais dans tous les cas, leur efficacité dépend du conditionnement du système. Le conditionnement se mesure grâce au nombre de conditionnement.

Définition 2.2.2.1 *(Nombre de conditionnement)*
Pour une matrice A, le nombre de conditionnement $\kappa(A)$ se définit par

$$\kappa(A) = \begin{cases} ||A||\ ||A^{-1}||, & \text{si } A \text{ est inversible,}\\ +\infty, & \text{sinon,}\end{cases}$$

où $||.||$ est une norme matricielle quelconque.

Lorsque $\kappa(A)$ est grand, on dit que le conditionnement est mauvais ou que la matrice est mal conditionnée. Pour un système linéaire, si A est mal conditionnée alors on dit que le système $Ax = b$ est mal conditionné. Dans la pratique, cela signifie qu'une légère variation des composantes de b entraîne une grande variation des composantes

de la solution x. Pour remédier à ce genre de problèmes, on peut *préconditionner* le système. Voyons en détails ce que cela signifie.

Définition 2.2.2.2 *(Préconditionneur)*
Soit A une matrice inversible de $\mathbb{R}^{n\times n}$ et b un vecteur de $\mathbb{R}^n$. Supposons qu'on cherche à résoudre le système $Ax = b$ et que ce système est mal conditionné. Un préconditonneur est alors une matrice M de $\mathbb{R}^{n\times n}$ symétrique définie positive telle que le système $MAx = Mb$ soit mieux conditionné que le problème original.

Dans le contexte des méthodes de régions de confiance, on cherche donc une matrice M telle que $MH_k s = -M\nabla f(x)$ soit mieux conditionné que le système original. Comme on l'a vu, on résout ensuite ce système préconditionné, par exemple, par une méthode de Gradient Conjugué.

On peut encore noter une propriété intéressante du préconditionnement dans le cadre de l'application aux méthodes de régions de confiance : on peut montrer (Conn, Gould, Orban, & Toint, 2000) que préconditionner le système revient à changer la norme qui définit les contours de la région de confiance. La norme utilisée est alors une norme de mise à l'échelle associée à la matrice M où M est telle que la matrice $\mathcal{K}(v)$ définie en (2.11) est définie positive.

$$\begin{bmatrix} M + J^T(x)\Theta(v)J(x) + E^T(x)\Xi(v)E(x) & J^T(x)\hat{\Theta}(v) & E^T(x)\hat{\Xi}(v) \\ \hat{\Theta}(v)J(x) & \Theta(v) + US^{-1} & 0 \\ \hat{\Xi}(v)E & 0 & \Xi(v) + VT^{-1} \end{bmatrix} \tag{2.11}$$

La région de confiance possède alors une forme qui reflète les propriétés du problème. Si M était H, la région serait exactement alignée avec les contours de la fonction barrière. Essayons donc de regarder cette propriété à la lumière du problème d'optimisation de départ.

Le problème initial possède, *a priori*, des contraintes d'égalité et d'inégalité. À la suite de la transformation élastique (voir chapitre 1, section 1.2.4) et lors de l'application de la méthode de points intérieurs, on se retrouve alors avec un problème d'optimisation sans contrainte (on doit minimiser la fonction barrière plus la pénalité, voir chapitre 1, section 1.2.5)). On peut alors utiliser un algorithme de région de confiance (voir chapitre 1, section 1.2.1)). D'après la propriété précédente, on comprend alors que si on choisit judicieusement M, on pourrait faire en sorte que la région

de confiance tienne compte des particularités du contour de la région admissible.

Illustration

Pour illustrer ces propos, considérons le cas à deux dimensions. Supposons pour simplifier l'explication que la norme associée à M, définie positive, est telle que $\forall d \in \mathbb{R}^n, ||d||_M = d^T M d$. On sait alors que pour $M \in \mathbb{R}^{2\times 2}$, les courbes de niveaux de $||d||_M$ sont des ellipses. De plus, on sait que les demi-axes sont parallèles aux vecteurs propres et leur longueur est inversement proportionnelle à la valeur propre associée.

Dans notre cas, prenons une matrice $M(x)$, qui dépend du point où on se trouve, dont les directions propres sont les directions des contraintes et telle que les valeurs propres sont proportionnelles à la distance entre x et les contraintes. Supposons que la contrainte 1 soit presque active. On obtient alors la figure 2.2.

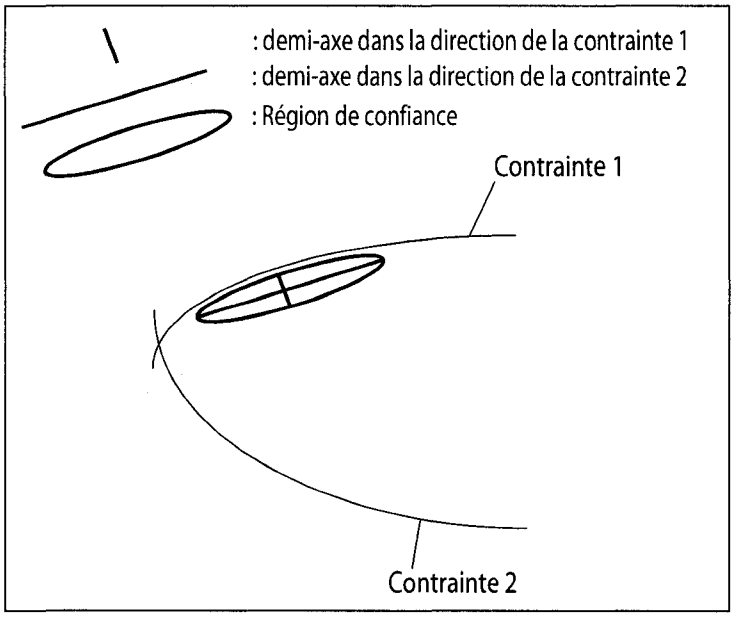


Figure 2.2 – Préconditionnement et région de confiance

La région de confiance est alors adaptée à la forme de la contrainte. Ainsi, imaginons le cas où un itéré se retrouve proche d'une frontière du domaine réalisable. Alors, grâce à la forme de la région de confiance, on est encore capable de chercher des pas dans de nombreuses directions qui vont par exemple nous permettre de progresser le

long de la contrainte. Remarquons que si la forme de la région de confiance n'est pas adaptée aux contraintes, on peut se retrouver dans des situations où on est bloqué sur la frontière alors que dans le cas contraire, on évite ce problème. Une illustration de ce phénomène est donnée à la figure 2.3.

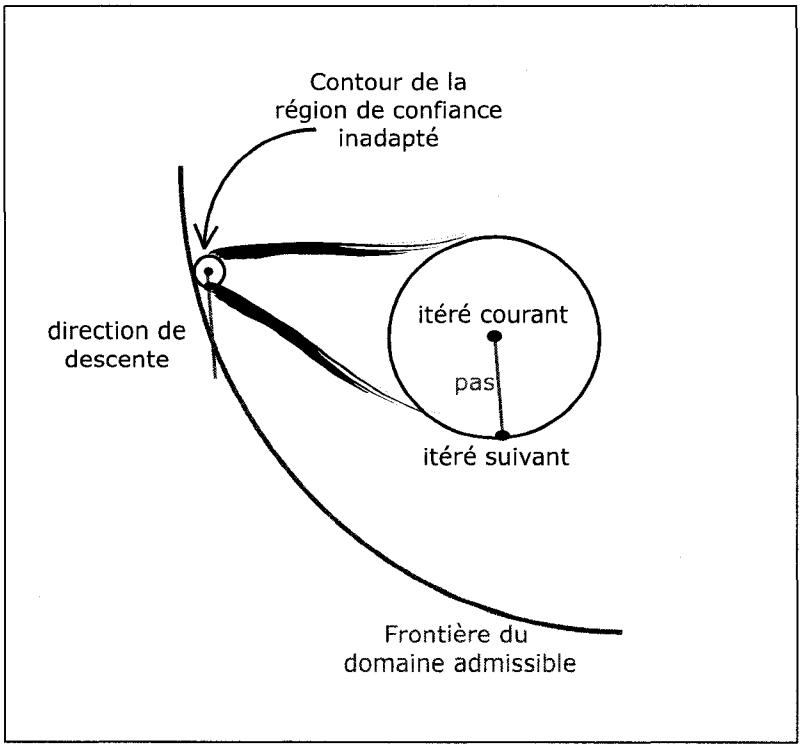


Figure 2.3 – Situation de bloquage sur une frontière

Dans la pratique, il faut choisir cette matrice M. A cette fin, remarquons la chose suivante. Nous savons que le Jacobien et le Hessien des contraintes nous donnent les informations nécessaires sur la forme des contraintes. On sait aussi que $\nabla_{xx}\mathcal{L}(x,\lambda)$ contient ces éléments. On sait aussi que proche d'une solution, les contraintes actives dominent dans $\nabla_{xx}\mathcal{L}(x,\lambda)$. En conséquence, les valeurs propres associées à ces directions dominent. Il est clair que la matrice $M = \nabla_{xx}\mathcal{L}(x,\lambda)$ a les propriétés souhaitées : quand on se rapproche d'une contrainte, la région de confiance adapte son contour à la forme de la contrainte en question. Remarquons que numériquement, il faut pouvoir

factoriser le préconditonneur et par conséquent, M doit être une matrice creuse. Le Hessien du Lagrangien est potentiellement une matrice dense, c'est pourquoi on ne le choisit pas en tant que préconditionneur. Une approximation creuse du Hessien du Lagrangien se révèle être une bonne option ainsi que nous le verrons dans la section sur les résultats numériques.

Revenons à présent sur le lien entre le préconditionneur et la résolution du sous-problème de région de confiance. Dans notre algorithme, on utilise une méthode de Gradient Conjugué préconditionné. Rappelons que la méthode du Gradient Conjugué est une méthode itérative pour résoudre les systèmes du type $Ax = b$ où A est définie positive. Or à une itération donnée de notre sous-problème de région de confiance, on doit résoudre

$$-\nabla\phi^B = \begin{bmatrix} H(x,\lambda(y,\nu)) + J^T(x)\Theta(v)J(x) + E^T(x)\Xi(v)E(x) & J^T(x)\hat{\Theta}(v) & E^T(x)\hat{\Xi}(v) \\ \hat{\Theta}(v)J(x) & \Theta(v) + US^{-1} & 0 \\ \hat{\Xi}(v)E & 0 & \Xi(v) + VT^{-1} \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_t \end{bmatrix}$$

Notre outil de résolution est donc la méthode du Gradient Conjugué. Mais on utilise la version préconditionnée de cette méthode (la version "non-préconditionnée" est en fait préconditionnée par l'identité). Ainsi on s'assure de la qualité des solutions trouvées. En contrepartie, cela implique qu'on résolve des sytèmes du type

$$\mathcal{K}(v)d = r$$

où $\mathcal{K}(v)$ est définie en (2.11) (notons que dans la version non-préconditionnée de l'algorithme on a $\mathcal{K}(v) = I$),

$$d = \begin{bmatrix} d_x \\ d_s \\ d_t \end{bmatrix}$$

et r un vecteur qui intervient au cours de la méthode. Comme on l'a dit plus haut, la matrice $\mathcal{K}(v)$ est définie positive. On dispose alors de moyens très efficaces (par le biais de factorisations) pour résoudre ce type de systèmes. Mais si on regarde de plus près $\mathcal{K}(v)$, on s'aperçoit alors que la matrice

$$J^T(x)\Theta(v)J(x)$$

pourrait poser des problèmes. En effet, il y a de fortes chances pour qu'elle soit dense et alors la factorisation de $\mathcal{K}(v)$ pourrait se révéler gourmande en espace mémoire puisqu'il faudrait stocker un grand nombre de termes. Pour éviter cet écueuil et suivant (Gould et al., 2003) nous avons introduit deux variables auxiliaires qui permettent d'éliminer les termes denses. On pose d'abord

$$\zeta = \Theta(v)J(x)d_x + \hat{\Theta}(v)d_s.$$

Le système se réecrit alors :

$$\begin{bmatrix} M + E^T(x)\Xi(v)E(x) & 0 & E^T(x)\hat{\Xi}(v) & J^T(x) \\ (\hat{\Theta}(v) - \Theta(x))J(x) & (\Theta(v) - \hat{\Theta}(v)) + US^{-1} & 0 & I \\ \hat{\Xi}(v)E & 0 & \Xi(v) + VT^{-1} & 0 \\ \Theta(v)J(x) & \hat{\Theta}(v) & 0 & -I \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_t \\ \zeta \end{bmatrix} = r$$

Le deuxième changement de variable est alors :

$$\xi = \Xi(v)E(x)d_x + \hat{\Xi}(v)d_t.$$

Le système peut se réécrire :

$$\begin{bmatrix} M & 0 & E^T(x)\hat{\Xi}(v) & J^T(x) & E^T(x) \\ (\hat{\Theta}(v) - \Theta(v))J(x) & (\Theta(v) - \hat{\Theta}(v)) + US^{-1} & 0 & I & 0 \\ (\hat{\Xi}(v) - \Xi(v))E & 0 & (\Xi(v) - \hat{\Xi}(v)) + VT^{-1} & 0 & I \\ \Theta(v)J(x) & \hat{\Theta}(v) & 0 & -I & 0 \\ \Xi(v)E(x) & 0 & \hat{\Xi}(v) & 0 & -I \end{bmatrix} d = r$$

où

$$d = \begin{bmatrix} d_x \\ d_s \\ d_t \\ \zeta \\ \xi \end{bmatrix} \text{ et } r = \begin{bmatrix} r_x \\ r_s \\ r_t \\ 0 \\ 0 \end{bmatrix}.$$

On élimine alors d_s et d_t et on obtient

$$\mathcal{K}(v)d \equiv \begin{bmatrix} M & J^T(x) & E^T(x) \\ D_1J(x) & B_1 & 0 \\ D_2E(x) & 0 & B_2 \end{bmatrix} \begin{bmatrix} d_x \\ \zeta \\ \xi \end{bmatrix} = \begin{bmatrix} r_x \\ C_1r_s \\ C_2r_t \end{bmatrix}$$

où :

$$D_1 = \Theta(v) - \hat{\Theta}(v)(\hat{\Theta} - \Theta)(\Theta - \hat{\Theta} + US^{-1})^{-1} \tag{2.12a}$$

$$B_1 = -I - \hat{\Theta}(v)(\Theta - \hat{\Theta} + US^{-1})^{-1} \tag{2.12b}$$

$$C_1 = -\hat{\Theta}(v)(\Theta - \hat{\Theta} + US^{-1})^{-1} \tag{2.12c}$$

$$D_2 = \Xi(v) - \hat{\Xi}(v)(\hat{\Xi} - \Xi)(\Xi - \hat{\Xi} + US^{-1})^{-1} \tag{2.12d}$$

$$B_2 = -I - \hat{\Xi}(v)(\Xi - \hat{\Xi} + US^{-1})^{-1} \tag{2.12e}$$

$$C_2 = -\hat{\Xi}(v)(\Xi - \hat{\Xi} + US^{-1})^{-1} \tag{2.12f}$$

Finalement, en multipliant la deuxième ligne par D_1^{-1} et la troisième par D_2^{-1} on aboutit au système symétrique

$$Pd \equiv \begin{bmatrix} M & J^T(x) & E^T(x) \\ J(x) & D_1^{-1}B_1 & 0 \\ E(x) & 0 & D_2^{-1}B_2 \end{bmatrix} \begin{bmatrix} d_x \\ \zeta \\ \xi \end{bmatrix} = \begin{bmatrix} r_x \\ D_1^{-1}C_1 r_s \\ D_2^{-1}C_2 r_t \end{bmatrix} \tag{2.13}$$

Remarque : D_1 et D_2 sont diagonales définies positives et B_1 et B_2 sont diagonales définies négatives (d'après les définitions (2.5), (2.6), (2.1.4.2) et (2.7)).
On note alors $D_1^{-1}B_1 = -F1$ et $D_2^{-1}B_2 = -F2$ avec $F1 \in \mathbb{R}^{nc \times nc}$ et $F2 \in \mathbb{R}^{n_B \times n_B}$ définie positives. On peut en déduire les conditions suivantes sur M pour s'assurer d'avoir un préconditionneur adapté c'est-à-dire pour que $\mathcal{K}(v)$ doit définie positive.

Théorème 2.2.1 *(Admissiblité d'un préconditionneur) Soit M un préconditionneur pour (2.9). On a alors*

$$\mathcal{K}(v) \textit{ définie positive} \iff M + J^T F_1^{-1} J + E^T F_2^{-1} E \textit{ définie positive}$$

Preuve : Soit M un préconditionneur et soit $\mathcal{K}$(v) la matrice de (2.11). On sait d'après (Gould, 1985) que

$$\mathcal{K}(v) \text{ définie positive}$$
$$\iff$$
la matrice P de (2.13) a exactement $n_C + n_B$ valeurs propres négatives.

Or on peut toujours décomposer P de la manière suivante

$$P = \begin{bmatrix} I & -J^T F_1^{-1} & -E^T F_2^{-1} \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} M + J^T F_1^{-1} J + E^T F_2^{-1} E & 0 & 0 \\ 0 & -F_1 & 0 \\ 0 & 0 & -F_2 \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ -F_1^{-1} J & I & 0 \\ -F_2^{-1} E & 0 & I \end{bmatrix}$$

et comme F_1 et F_2 sont définies positives de tailles respectives $n_{\mathcal{C}} \times n_{\mathcal{C}}$ et $n_{\mathcal{B}} \times n_{\mathcal{B}}$, le résultat est immédiat. □

2.2.3 Méthode de Nocedal-Yuan

Dans leur article (Nocedal & Yuan, 1998) proposent un algorithme d'optimisation non-linéaire qui utilise à la fois les méthodes de région de confiance et de recherche linéaire. En effet, dans leur algorithme, lorsque le pas d'essai donné par la région de confiance ne permet pas de faire décroître l'objectif, au lieu de le rejeter (méthodes classiques) on procède plutôt à une recherche linéaire le long de la direction trouvée. Ce qui justifie la méthode est assez simple à concevoir. En effet, dans les méthodes de région de confiance classiques, on sait ce qui arrive :

- On fait décroître un modèle itérativement dans la région de confiance. On obtient ainsi un pas.
- On rejette ce pas s'il ne produit pas une décroissance suffisante de l'objectif.

Dans le cas où le pas est rejeté, on réduit alors le rayon de la région de confiance et on recommence. Lorsqu'on utilise une méthode itérative pour obtenir la décroissance du modèle, les premières itérations de ce procédé sont toujours les mêmes, quel que soit le rayon de la région de confiance. Voyons cela sur un exemple.

Exemple

Supposons qu'on soit dans un problème à deux dimensions et que l'itéré courant soit x_k. On cherche alors à résoudre itérativement

$$\begin{aligned} \underset{s}{\text{minimiser}} \quad & q(s) \\ \text{s.c :} \quad & ||s|| \leq \Delta \end{aligned}$$

où q est le modèle quadratique, $s \in \mathbb{R}^2$ est le pas, $||.||$ est la norme euclidienne et Δ le rayon de la région de confiance actuelle. Supposons qu'on utilise une méthode de *Gradient conjugué* et que $\Delta = \Delta_2$. La figure 2.4 montre alors la solution obtenue avec

cette méthode lorsque $\Delta = \Delta_2$. Si la solution obtenue ne produit pas une décroissance suffisante, on rejette le pas obtenu, p, et on réduit le rayon qui vaut alors Δ_1. On recommence le procédé et on voit que la première direction de *Gradient conjugué* calculée, P_1, est la même qu'à l'étape précédente lorsque Δ valait Δ_2. On répète ainsi des calculs déjà effectués.

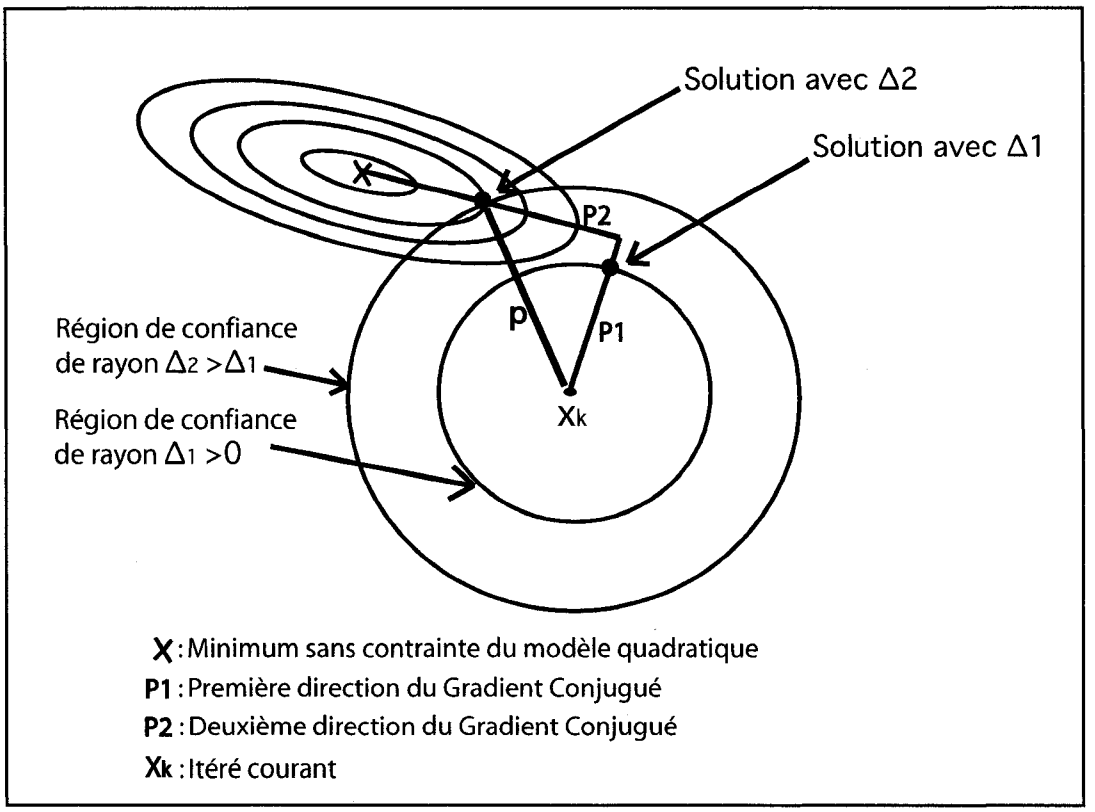


Figure 2.4 – Méthode itérative et région de confiance

Lorsqu'on s'intéresse à des problèmes de petite taille, la répétition des calculs des directions de *Gradient conjugué* affecte peu les performances d'un algorithme mais cela devient problématique dans le cas où on traite des problèmes de grande taille. Notons que lorsqu'un pas p est rejeté, il constitue quand même une direction de descente. L'idée proposée par Nocedal et Yuan est donc de procéder directement à une *recherche linéaire* le long cette direction de descente.

Il est aussi montré dans l'article que les propriétés de convergence des méthodes de région de confiance sont conservées. Plusieurs algorithmes (plus ou moins raffinés) sont présentés dans l'article et nous avons retenu l'algoritme 3.1 pour des raisons de simplicité d'intégration au code déjà écrit et pour son apparente efficacité (voir les résultats numériques dans l'article).

Pour établir le lien avec ce que nous avons déjà vu, comparons cet algorithme à l'algorithme 1.2.1. Il n'y a qu'une seule grande différence et elle se situe au niveau de l'étape 2. On avait :

Etape 2. Acceptation du point d'essai - Calculer $f(x_k + s_k)$ et

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}$$

Si $\eta_1 \leq \rho_k$ alors poser $x_{k+1} = x_k + s_k$;
Sinon poser $x_{k+1} = x_k$.

alors qu'avec la version de Nocedal et Yuan on a :

Etape 2. N-Y Acceptation du point d'essai - Calculer $f(x_k + s_k)$ et

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}$$

Si $\eta_1 \leq \rho_k$ alors poser $x_{k+1} = x_k + s_k$;
Sinon faire une recherche linéaire le long de s_k
et poser $x_{k+1} = x_k + \gamma s_k$ **où** $\gamma > 0$ **est issu de la recherche linéaire.**

Une fois modifié, l'algorithme 2.1.1 devient l'algorithme 2.2.1. Nous avons décidé d'étudier l'impact numérique de cette option lorsqu'elle est implémentée avec PSUPERB. Nous présentons les résultats de cette étude dans le chapitre 3.

Algorithme 2.2.1 PSUPERB - NY Implémenté

Étape 0. (Initialisation)

Choisir des valeurs initiales x^0, s^0, t^0, ν^0 , μ^0 et Δ^0.
Choisir des constantes $\gamma_1 > 1 > \gamma_2 > 0$ et $\Delta_{max} > 0$.
Poser $k = 0$.

Étape 1. (SolveOuterIteration)

Si les conditions de KKT pour (NLP) ne sont pas satisfaites :
incrémenter k ;
mettre à jour ν ;
mettre à jour μ ;
aller à Étape 2.
Sinon : STOP.

Étape 2. (SolveInnerIteration)

Si les conditions (1.39) ne sont pas satisfaites :
- aller à Étape 3.
Sinon : aller à Étape 1.

Étape 3. (SolveSubproblem)

Résoudre une itération de sous-problème barrière grâce à TRPCG.
Obtenir un pas d.
Si d produit une décroissance suffisante de la fonction barrière :
-aller à Étape 4.
Sinon : aller à Étape 5.

Étape 4. (Mise à jour positive de la Région de Confiance)

Mettre à jour x^k, s^k, t^k.
On augmente le rayon de la région de confiance : $\Delta^k = min(\gamma_1 \Delta^k, \Delta_{max})$.
Aller à Étape 2.

Étape 5. (Mise à jour négative de la Région de Confiance)

On diminue le rayon de la région de confiance : $\Delta^k = \gamma_2 \Delta^k$.
On fait du backtracking sur le pas d.
On met à jour x^k, s^k, t^k .
Aller à Étape 2 .

Chapitre 3

Résultats Numériques

Dans ce chapitre, on s'attache à l'aspect numérique de notre algortihme. On propose d'abord une étude de quelques problèmes de référence, simples, sur lesquels on peut comprendre ce qui se passe, étape par étape. Ensuite, on présente les résultats des tests numériques que nous avons effectués. Précisons d'ores et déjà que nous avons produit trois variantes de PSUPERB : la variante de base, celle avec un préconditionnement (voir section 2.2.2 du chapitre 2) et celle avec une méthode de Nocedal et Yuan (voir section 2.2.3 du chapitre 2).

3.1 Problèmes de référence

Dans ce paragraphe, nous allons nous essayer de comprendre comment fonctionne PSUPERB sur trois problèmes relativement simples mais qui illustrent différents comportements de l'algorithme.

Problème 1

Ce problème est tiré de la collection (Hock & Schittkowski, 1981). Son modèle est le suivant :

$$\begin{array}{ll} \underset{(x_1,x_2)}{\text{minimiser}} & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ \text{s.c. :} & -1,5 \leq x_2, \end{array}$$

et le point de départ est un point admissible : $x_1 = -2$ et $x_2 = 1$. Essayons de voir comment PSUPERB transforme ce problème. D'abord on a :

$$\phi^S(x_1, x_2, t, \nu) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + \nu t$$

et le problème associé est :

$$\begin{aligned} \underset{(x_1,x_2,t)}{\text{minimiser}} \quad & \phi^S(x_1, x_2, t, \nu) \\ \text{s.c. :} \quad & 0 \leq x_2 + 1,5 + t, \\ & 0 \leq t. \end{aligned}$$

Ensuite on introduit :

$$\phi^B(x_1, x_2, t, \nu, \mu) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + \nu t - \mu \log(x_2 + 1,5 + t) - \mu \log(t)$$

et on doit résoudre le sous-problème barrière (sans contraintes)

$$\text{minimiser} \quad \phi^B$$

pour chaque valeur fixée de μ et de ν. Pour cela il faut satisfaire les conditions (1.39) pour ce sous-problème. La solution exacte du sous-problème barrière est la suivante (pour les variables primales) :

$$\begin{bmatrix} x_1 \\ x_2 \\ t \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \mu/\nu. \end{bmatrix}.$$

Quand on fait tourner PSUPERB, on aboutit au résultat présenté dans le tableau 3.1 pour les solutions du sous-problème barrière.

Tableau 3.1: Suite des itérés proposée par PSUPERB sur un problème non-linéaire

Itération	x_1	x_2	t	μ	ν
0	0.894	0.798	1.373	0.2	3
1	1.047	1.096	0.229	0.04	3
2	1.016	1.032	0.041	0.008	3
3	1.003	1.006	0.008	0.0016	3
4	1.000	1.001	0.001	0.00032	3
5	1.0001	1.0002	3e-04	6.4e-05	3
6	1.00002	1.00005	6.4e-05	1.28e-05	3
7	1.000005	1.00001	1.27e-05	2.56e-06	3
8	1.000001	1.000002	2.56e-06	5.12e-07	3

On remarque que ν reste borné, que μ tend vers 0 et par conséquent, conformément à ce que donne la solution exacte, la grandeur t tend vers 0. Pour x_1 et x_2, les itérés donnés par l'algorithme convergent bien vers ceux proposé par la théorie.

Les deux autres exemples sont des problèmes pour lesquels la MFCQ n'est pas satisfaite au point optimal. Les comportements de PSUPERB peuvent alors être différents.

Problème 2

Ce problème est aussi tiré de la collection (Hock & Schittkowski, 1981) (HS 013) et son modèle a été présenté dans le chapitre 1. Ce problème présente la particularité d'avoir un point optimal pour lequel la MFCQ n'est pas satisfaite. Sur un tel problème, PSUPERB n'est pas supposé aboutir à une solution qui lui semble optimale à moins qu'on n'utilise la règle (1.2.6.1) de mise à jour du paramètre de pénalité. Son comportement est le suivant si on utilise la règle classique (1.2.5.4) : il se dirige bien vers le bon point mais n'arrivant pas à satisfaire les conditions de KKT (*ie* les conditions (1.35) pour (1.29)), il ne renvoie pas un message d'optimalité. On notera que la précision avec laquelle il approche le point optimal dépend fortement du ν initial choisi comme le montre le tableau 3.2.

Tableau 3.2: Variation du point d'adhérence en fonction du paramètre de pénalité initial

ν initial	ν final	x_1	x_2	Objectif
(10, 10, 10)	(21870, 21870, 21870)	1.22	-1.4e-4	0.59
(100, 100, 100)	(112010, 112010, 112010)	1.03	1.3e-6	0.85
(1000, 1000, 1000)	(1000, 1000, 1000)	1.08	1.2e-7	0.95
(10000, 10000, 10000)	(40000, 40000, 40000)	1.004	-3.1e-8	0.99

On voit clairement que les résultats varient beaucoup et qu'avec certains ν initiaux on peut se rapprocher fortement du point optimal (dernière ligne du tableau 3.2). Quand on observe bien le tableau précédent, on pourrait être tenté de dégager une évolution générale de la qualité des résultats qui irait dans le sens suivant : «plus le ν initial est grand, plus on se rapproche de la solution».

Il est alors temps de faire le lien entre ces observations et le théorème 1.2.5 du chapitre 1. En effet, ce théorème peut se résumer ainsi : si la suite de ν générés par l'algorithme tend vers l'infini et que PSUPERB converge vers un point x^* admissible

alors la MFCQ n'est pas vérifiée en ce point et c'est un point de Fritz-John du problème de départ.

Dans le problème considéré, on sait (voir section 1.1.3) que la MFCQ n'est pas vérifiée au point optimal (1, 0). On voit aussi qu'un grand ν semble approprié pour s'approcher du point (1,0), admissible (la solution). On est amené à penser que le point optimal est un point de Fritz-John du problème de départ. Pour le prouver, nous avons implémenté les conditions d'optimalité de Fritz-John afin de voir si on pouvait les vérifier numériquement. Concrètement, nous avons demandé à l'algorithme de vérifier ces conditions (au lieu des conditions de KKT) si le point en lequel on se trouve est admissible à 10^{-6} près et que ν est supérieur à 10000. Un test numérique où on prend le ν initial tel que $\nu = (6000, 3000, 6000)$ donne un résultat positif en 45 itérations. C'est-à-dire que PSUPERB converge vers le point (1,0) admissible avec un ν qui semble tendre vers l'infini (on termine avec $\max(\nu_{Eq}, \nu_S, \nu_T) = 24000$).

Problème 3

Ce problème est issu de la collection MacMPEC (voir (Leyffer, 2005)).
Le modèle est le suivant :

$$\begin{aligned} \underset{(x_1,x_2)}{\text{minimiser}} \quad & x_1^2 + x_2^2 - 4x_1x_2 \\ \text{s.c. :} \quad & x_1x_2 = 0, \\ & 0 \leq x_1, \\ & 0 \leq x_2. \end{aligned}$$

Le point optimal pour ce problème est (0,0). Il est clair qu'au point optimal, le gradient de la contrainte est nul. La MFCQ n'est donc pas satisfaite (on le savait puisqu'on a démontré au chapitre 1 que la MFCQ n'est satisfaite en aucun point admissible d'un MPEC). Mais c'est un point fortement stationnaire. Comme on sait que PSUPERB peut converger vers un point fortement stationnaire (voir (Coulibaly, 2008)), il devrait théoriquement être en mesure de trouver la solution. La suite des itérés est présentée dans le tableau 3.3. Ces résultats numériques viennent appuyer les preuves théoriques concernant notre algorithme .

Tableau 3.3: Suite des itérés proposée par PSUPERB sur un problème de type MPEC

Itération	x_1	x_2
0	0.634	0.634
1	0.114	0.114
2	0.071	0.071
3	0.032	0.032
4	0.019	0.019
5	0.007	0.007
6	0.003	0.003
7	0.0017	0.0017
5	0.0006	0.0006

3.2 Tests numériques

3.2.1 Influence du paramètre de pénalisation

Nous présentons ici les résultats obtenus quant aux performances de PSUPERB en fonction de l'heuristique choisie pour calculer le ν initial. La collection de problèmes dont nous nous sommes servi est issue de (Hock & Schittkowski, 1981). Les tests ont été effectués sur 110 problèmes de cette collection. Le tableau 3.4 explique pourquoi certains problèmes n'ont pas été traités. Tous les problèmes sont non-linéaires, que ce soit par leur objectif ou leurs contraintes.

Tableau 3.4: Problèmes de la collection HS non-traités

Problèmes abandonnés	Cause
58, 82, 94, 115	Modèle inexistant
68, 69	Problème non différentiable

Afin d'analyser les performances de notre algorithme, nous avons utilisé un outil qui s'appelle un *profil de performance* développé dans (Dolan & Moré, 2001). C'est un outil qui permet de comparer divers algortihmes en termes de temps CPU, nombre d'itérations, etc... . Nous proposons au lecteur un rappel sur le fonctionnement de cet outil.

Pour un critère donné, la théorie est fondée sur le calcul de ratios : pour chaque algorithme et pour chaque problème, on calcule le rapport de la performance obtenue

par l'algorithme en question sur la meilleure performance. L'allure typique d'un profil de performance pour une certaine mesure de performance est donnée par la figure 3.1. Sur l'axe des ordonnées, on lit le pourcentage cumulé de problèmes résolus par un

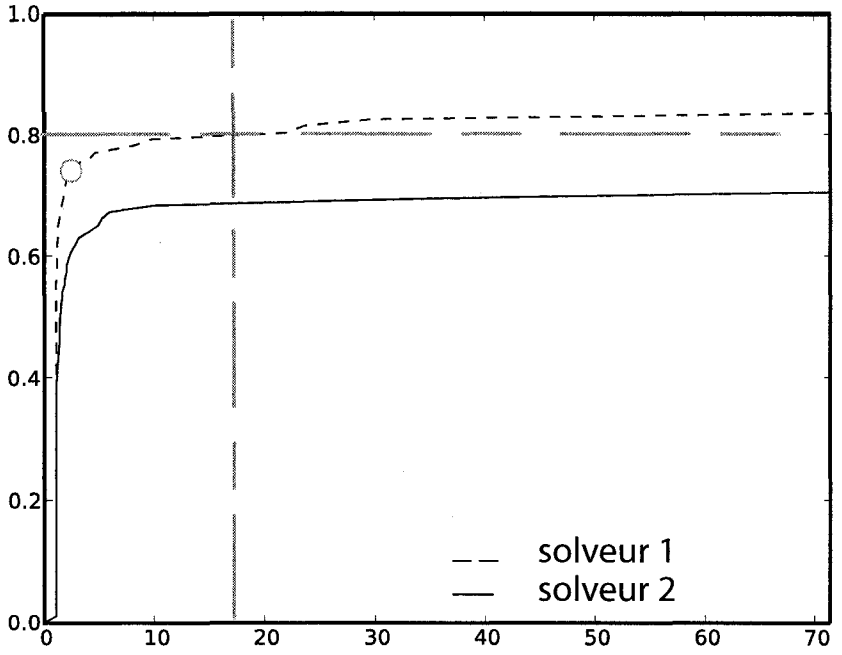


Figure 3.1 – Un profil de performance typique

solveur avec une efficacité inférieure ou égale à τ fois celle du meilleur solveur. On lit τ sur l'axe des abscisses (pour des raisons de lisibilité, on utilise souvent une échelle logarithmique). Considérons le *point rouge* par exemple. Il a pour abscisse 1 et pour ordonnée 75. Cela signifie que le solveur 1 produit la meilleure mesure de performance sur 75% des problèmes. Autrement dit, si on prend un problème parmi la collection de problèmes testés, la probabilité que le solveur 1 soit le meilleur est de 75%. La *ligne horizontale* nous indique quel est le solveur le plus apte à résoudre 80% des problèmes. Lorsqu'on part de la gauche pour aller vers la droite, les solveurs les plus efficaces sont les premiers rencontrés. Dans notre cas, le solveur 1 est plus apte que le solveur 2 à résoudre 80% des problèmes. La *ligne verticale* est utile quand on veut savoir la chose suivante : lorsqu'on s'autorise à être entre 1 et τ ($\tau = 17$ dans notre

figure) fois moins performant que le meilleur solveur, quel pourcentage de problèmes solutionne-t-on ? Sur l'exemple, le solveur 2 résout alors environ 70% des problèmes alors que le solveur 1 en résout plus de 80%. Enfin, les valeurs lues pour la plus grande valeur de τ donnent le pourcentage total de problèmes résolus par chaque solveur. Sur la figure, le solveur 2 résout un peu plus de 70% des problèmes alors que le solveur 1 atteint presque les 85%.

Globalement, si une courbe est **toujours au-dessus** d'une autre, cela signifie que l'algorithme qu'elle représente est **plus performant** que l'autre. Pour terminer, notons que dans le cas où les courbes se croisent, l'interprétation est plus difficile. En effet, cela signifie qu'un solveur est meilleur que l'autre si on considère une performance de 1 à 7 (par exemple) fois moins bonne que la performance optimale mais la tendance s'inverse si on considère une autre fourchette de performance. Il est alors difficile de distinguer clairement un *meilleur* solveur. On peut à présent disposer de cet outil pour comparer nos diverses versions de PSUPERB ainsi que les performances de l'algorithme pour deux manières différentes de calculer le ν initial.

Notation 3.2.1.1 *Versions de PSUPERB*
On note les différentes versions de PSUPERB de la façon suivante :

Version1, *la version de base*

Version2, *la version préconditionnée*

Version3, *le version avec la méthode de Nocedal et Yuan.*

Pour les heuristiques utilisées, la notation suit ce qui a été développé dans la section 2.2.1 : l'heuristique 1 est fondée sur le calcul du gradient de l'objectif au point initial et l'heuristique 2 sur les conditions de KKT pour le sous-problème barrière.

Pour évaluer les performances des heuristiques, on a comparé les résultats obtenus en termes de temps CPU avec les deux choix possibles du ν initial, pour une version de PSUPERB donnée. Le temps CPU semblait un bon critère pour mettre en parallèle les résultats puisque tous les tests ont été effectué sur la même machine et que cela reste un critère industriel intéressant. On obtient les figures 3.2, 3.3 et 3.4. **Interprétation** : Pour les versions 1 et 2, on s'aperçoit que les courbes se croisent, parfois plusieurs fois et qu'elles sont très proches l'une de l'autre. A la fin, on résout environ le même pourcentage de problèmes. Les performances entre

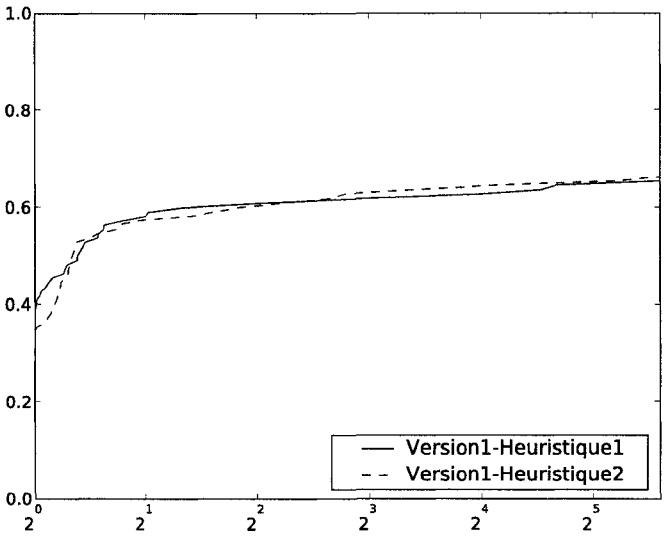


Figure 3.2 – Comparaison des heuristiques pour la Version 1 (Temps CPU)

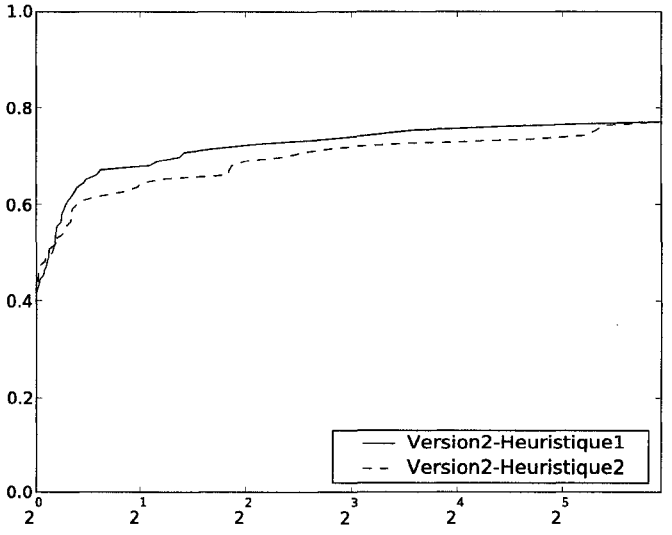


Figure 3.3 – Comparaison des heuristiques pour la Version 2 (Temps CPU)

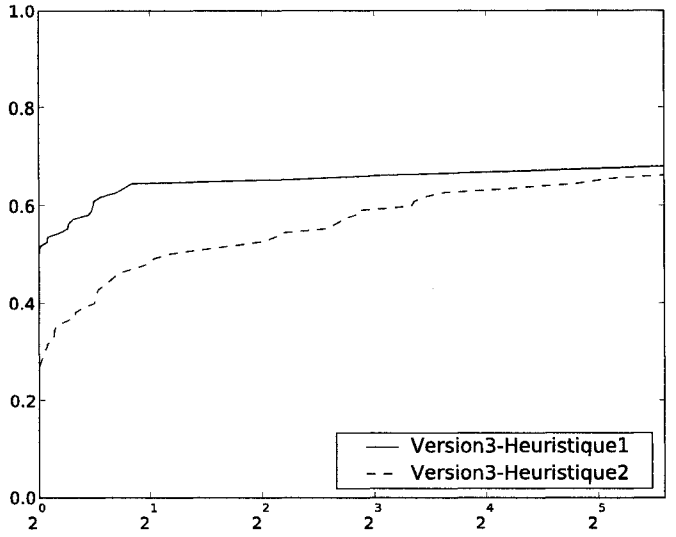


Figure 3.4 – Comparaison des heuristiques pour la Version 3 (Temps CPU)

les deux heuristiques sont donc très semblables et on ne peut pas vraiment déclarer qu'une heuristique est meilleure que l'autre. La différence est plus lisible sur le graphe (3.4) où la courbe correspondant à l'heuristique 2 est toujours au-dessus de l'autre. Cependant, à la fin, le pourcentage de problèmes résolus est quasiment le même.
Conclusion : les performances des heuristiques sont très semblables et on peut donc aussi bien utiliser l'une que l'autre même si l'heuristique 1 possède un léger avantage lorsqu'on emploie la Version 3.

3.2.2 Impact du préconditionnement et de la méthode de Nocedal-Yuan

A présent, il s'agit de comparer les différentes versions entre elles. Pour cela on fixe l'heuristique et on dresse le profil de performance pour les critères du temps CPU et du nombre d'itérations pour les trois algorithmes.

3.2.2.1 Comparaison des temps CPU

Pour le critère du temps CPU, on obtient les figures 3.5 et 3.6.

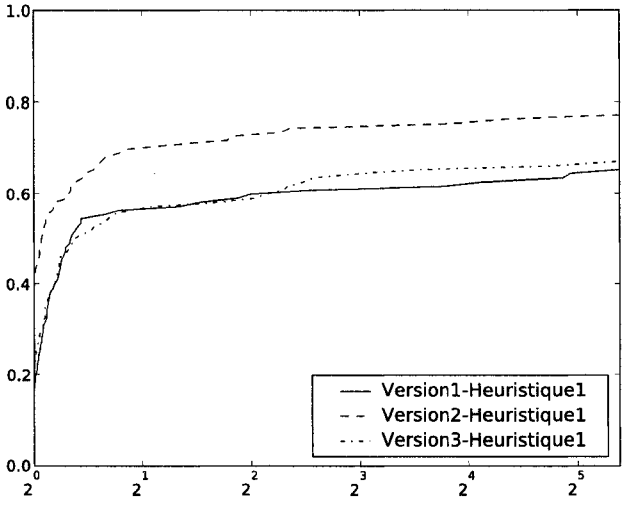


Figure 3.5 – Comparaison des versions pour l'heuristique 1 (Temps CPU)

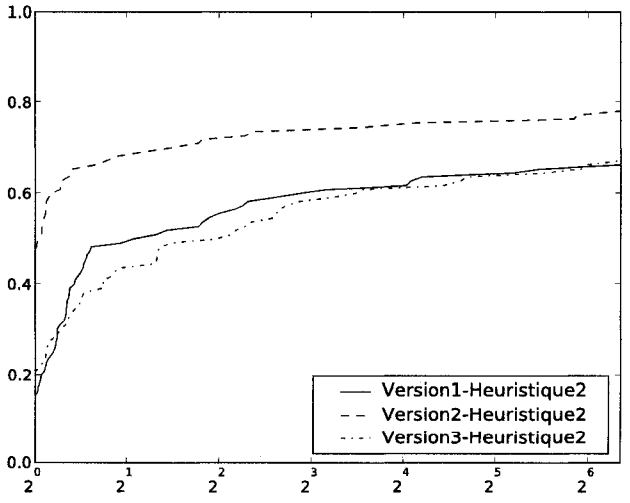


Figure 3.6 – Comparaison des versions pour l'heuristique 2 (Temps CPU)

Interprétation : les résultats sont plus clairs que dans les figures (3.2)-(3.4). La courbe correspondant à la Version 2 est, dans les deux cas, au-dessus des autres. La Version 2 a généré une meilleure solution (c'est-à-dire a été plus rapide puisqu'on mesure le temps CPU) sur environ 45% (respectivement 50%) des problèmes avec l'heuristique 1 (respectivement l'heuristique 2). De plus, la Version 2 résout presque 80% des problèmes dans les deux cas alors que les autres versions tournent autour de 65%. Si on cherche maintenant à comparer les deux autres versions entre elles, la Version 1 est supérieure à la Version 3 même si cette supériorité n'est pas évidente. En effet, bien que la courbe correspondant à la Version 1 soit plus souvent au-dessus l'autre, les deux courbes se croisent. En termes de probabilité de résoudre un problème pris au hasard, les deux versions sont presque identiques, ainsi que pour le pourcentage de problèmes résolus.

Conclusion : En termes de temps CPU, la Version 2 est donc globalement meilleure que les deux autres qui, elles, sont très semblables.

3.2.2.2 Comparaison des nombres d'itérations

On compare dans ce qui suit les nombres d'itérations internes nécessaires à la résolution du problème. On obtient alors les figures 3.7 et 3.8

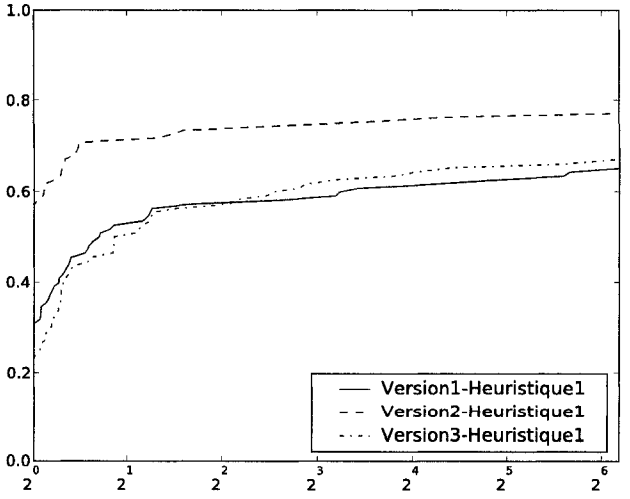


Figure 3.7 – Comparaison des versions pour l'heuristique 1 (Nombre d'itérations)

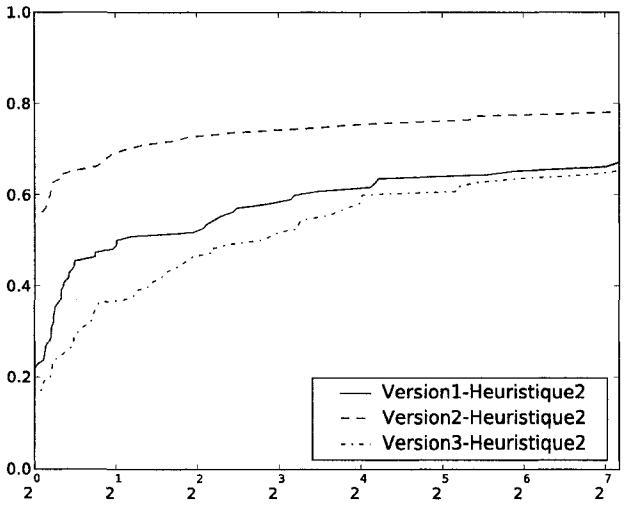


Figure 3.8 – Comparaison des versions pour l'heuristique 2 (Nombre d'itérations)

Interprétation : les résultats sont encore plus marqués que précédemment. En effet, les trois courbes sont disposées les unes au-dessus des autres : celle correspondant à la Version 2 au-dessus de celle correspondant à la Version 1, elle-même au-dessus de celle correspondant à la Version 3. On remarque que pour un problème pris aléatoirement, la Version 2 a cette fois presque 60% de chances d'être la meilleure en termes de nombre d'itération.

Conclusion : En termes de nombre d'itérations, la Version 2 est donc globalement meilleure que les deux autres et la Version 1 est meilleure que la Version 3. Si on veut résoudre un problème en cherchant à minimiser le nombre d'itérations, la Version 2 est donc celle qu'il faut choisir.

3.2.3 Analyse

Nous essayons dans cette partie d'éclairer les résultats numériques précédents à la lumière de ce que l'aspect théorique nous apprend.

Supériorité de la Version 2 :

Ainsi que nous l'avons vu dans la section 2.2.2, on sait le préconditionnement apporte une stabilité numérique aux dépens d'un nombre accru de calculs à effectuer (car on

sait qu'on doit résoudre des systèmes linéaires supplémentaires). On ne peut donc pas prévoir, *a priori*, si l'aspect bénéfique l'emportera numériquement sur l'aspect négatif. On peut notamment imaginer que le temps de calcul sera plus grand dans le cas d'un préconditionnement à cause des calculs supplémentaires.

La pratique nous montre donc que c'est l'aspect positif du préconditionnement qui remporte le duel. En effet, grâce à l'amélioration de la stabilité numérique, on diminue fortement le nombre d'itérations nécessaires et au final, le temps de calcul est moins élevé. Cet enchaînement de bénéfices n'était pas une évidence avant les tests numériques.

Échec relatif de la méthode de Nocedal et Yuan :
Dans la théorie, on a vu que la méthode de Nocedal et Yuan permettait d'éviter de répéter certains calculs lorsqu'on résout le sous-problème barrière par une méthode de région de confiance. Elle devrait donc à la fois réduire le nombre d'itérations pour résoudre le sous-problème et par conséquent le temps de résolution. Or dans les tests numériques, il apparaît que ce n'est pas le cas.

D'abord, il faut comprendre que la méthode a été conçue pour des problèmes de grande taille, pour lesquels le calcul de la direction de descente est très coûteux. La collection considérée ne possède pas vraiment ce genre de problème.

Ensuite, une manière peut-être plus adaptée de comparer les versions 1 et 3 seraient de le faire sur les problèmes que la Version 1 résout en un nombre élevé d'itérations. Ainsi, on pourrait voir si la méthode permet de gagner en efficacité sur les problèmes longs à résoudre et par conséquent l'utiliser dans ce cas uniquement. Quand on effectue ce travail, on est confronté à deux cas principaux :

Cas 1 Le nombre d'itérations avec la Version 3 est largement inférieur à celui de la Version 1. Exemple : problème hs078 où on passe de 857 à 152 itérations.

Cas 2 Le nombre d'itérations avec la Version 3 est comparable à celui de la Version 1. Exemple : problème hs059 pour lequel on passe de 175 à 146 itérations.

On s'aperçoit que la méthode de Nocedal et Yuan peut soit être très avantageuse, soit être presque inutile dans le cas de nos problèmes qui, il faut le rappeler, ne sont pas de grande taille. Notons qu'à l'avenir, on pourrait essayer de combiner le préconditionnement avec cette méthode de Nocedal et Yuan.

3.2.4 Résultats sur des MPEC

On propose ci-dessous l'évaluation des performances de la Version 2 de PSUPERB sur une collection de problèmes de type MPEC (voir (MPEC)). Cette collection est disponible sur le site web `www-unix.mcs.anl.gov/~leyffer/MacMPEC/` et propose des problèmes d'origine et de taille différentes. Nous avons sélectionné un ensemble de 63 problèmes parmi tous ceux qui étaient disponibles. Le tableau 3.5 récapitule les problèmes qui n'ont pas été étudiés. Nous donnons les résultats obtenus en fonction

Tableau 3.5: Problèmes de la collection MacMPEC non-traités

Problèmes abandonnés	Cause
design-cent-21, design-cent-3, design-cent-31, jr2 gnash11 à gnash19, pack-comp2*, gnash11m à gnash19m, incid-set1-16, incid-set1-32, incid-set1c-8, incid-set1c-16, incid-set1c-32, incid-set2-*, liswet-100, liswet-200, pack-comp1-8 à pack-comp1-32, pack-rig* (sauf pack-rig1-8). qpec* (sauf qpec1-100).	Modèle similaire à un problème déjà traité
bem-milanc30, siouxfls	Temps de calcul nécessaire trop élevé à cause de la taille du problème

des heuristiques combinées avec la Version 2 de l'algorithme à la figure 3.9. Remarquons que chaque modèle étudié a du être modifié manuellement pour remplacer le mot-clé `complements` du langage AMPL par les contraintes explicites qu'il représente car notre code ne reconnaît pas encore ce mot-clé.

Interprétation : on s'aperçoit (en regardant ce qui se passe lorsque l'abscisse est minimum) que les deux heuristiques donnent des résultats très similaires. En effet, si on prend un problème au hasard on a environ 50% de chance pour qu'il soit résolu plus vite par l'heuristique 1 (et donc le même résultat pour l'heuristique 2). Ensuite, on voit que les deux méthodes aboutissent à la résolution d'un peu moins de 80% des problèmes. Finalement on voit (en regardant ce qui se passe lorsque l'abscisse est maximum) qu'au pire, la meilleure heuristique est 16 fois plus rapide que l'autre. On

conclut donc que les deux heuristiques sont comparables en termes de temps CPU de résolution.

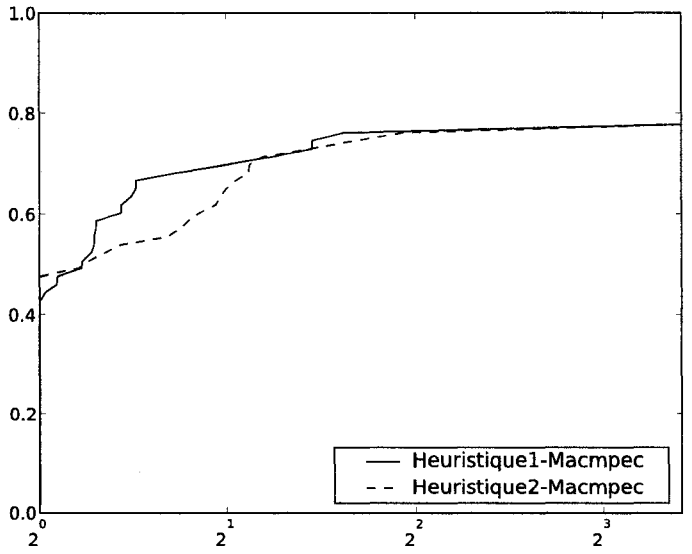


Figure 3.9 – Comparaison des performances (Temps CPU) des heuristiques avec la Version 2 sur la collection MacMPEC

Remarque : Lors des tests numériques, nous avons remarqué la chose suivante : sur certains problèmes de type MPEC, l'algorithme converge vers un point non-optimal (et par conséquent échoue) pour lequel *seule* la contrainte de type

$$(F^T x) \perp (G^T x) \tag{3.1}$$

est très mal satisfaite. Or la qualité de la satisfaction des contraintes est étroitement liée à la gestion des variables élastiques et donc au paramètre ν. Cela nous a conduit à imaginer le scénario suivant : si au cours de la boucle externe on remarque que les contraintes de type (3.1) sont mal satisfaites, on augmente alors le paramètre ν.

Cette approche est comparable à celle présentée dans (Leyffer, López-Calva, & Nocedal, 2006) dans lequel une mise à jour dynamique de ν est aussi évoquée. Après implémentation de cette stratégie, nous avons testé numériquement l'impact du changement. Voici trois exemples sur lesquels la modification a apporté un net bénéfice.

Exemple 1 : (scale)
Le problème est le suivant :

$$\begin{array}{ll} \underset{x_1,x_2\in\mathbb{R}}{\text{minimiser}} & (10x_1-1)^2+(1000x_2-1)^2 \\ \text{s.c. :} & x_1 \geq 0, \\ & x_2 \geq 0, \\ & x_1x_2 = 0. \end{array}$$

Lorsqu'on n'utilise pas la stratégie mentionnée plus haut, l'algorithme se dirige vers le point (0.1, 0.001) qui n'est pas admissible car la condition $x_1x_2 = 0$ n'est pas satisfaite car elle vaut 10^{-4} en ce point. Pourtant, ce point est optimal pour les problèmes élastiques successifs générés par l'algorithme car on peut trouver numériquement des valeurs pour les multiplicateurs afin qu'il en soit ainsi.
D'un autre côté, avec la mise à jour dynamique de ν, on converge vers le point optimal (0.1, 0) en 34 itérations l'heuristique 1 (693 avec l'heuristique 2). Cela provient du fait qu'on est très exigeant sur la satisfaction de la contrainte $x_1x_2 = 0$: au bout d'un moment, la valeur de 10^{-4} ne nous satisfait pas assez, ce qui nous conduit à augmenter ν et l'algorithme se dirige vers un autre point pour lequel la contrainte est mieux vérifiée. La règle exacte pour la mise à jour dynamique est la suivante :

$$\text{Si } ||x_1x_2|| \leq 10\mu \text{ Alors } [\nu_{Eq}, \nu_S, \nu_T] \leftarrow [10\nu_{Eq}, 10\nu_S, 10\nu_T].$$

Exemple 2 : (ralph1)
Le problème est le suivant :

$$\begin{array}{ll} \underset{x_1,x_2\in\mathbb{R}}{\text{minimiser}} & 2x_1 - x_2 \\ \text{s.c. :} & x_2 - x_1 \geq 0, \\ & (x_2 - x_1)x_2 = 0, \\ & x_1, x_2 \geq 0. \end{array}$$

Le point optimal pour ce problème est $(0, 0)$. Si on n'utilise pas la stratégie de mise à jour dynamique, on converge vers $(0, 0.01)$ qui encore une fois ne satisfait pas la contrainte $(x_2 - x_1)x_2 = 0$. Mais dans le cas où on emploie la nouvelle stratégie, on converge vers l'optimum en 34 itérations avec l'heuristique 1 (29 avec l'heuristique 2).

Exemple 3 : (scholtes4)

$$\begin{array}{ll} \underset{x_1,x_2\in\mathbb{R}}{\text{minimiser}} & x_1 + x_2 - x_3 \\ \text{s.c. :} & x_2 \geq 0, \\ & x_1 \geq 0, \\ & x_1 x_2 = 0, \\ & -4x_1 + x_3 \leq 0, \\ & -4x_2 + x_3 \leq 0. \end{array}$$

Le point optimal est $(0, 0, 0)$. On est face au même genre de situation que dans les deux exemples précédents :

- Sans mise à jour dynamique, on converge vers $(0.01, 0.01, 0.01)$. La contrainte de complémentarité $x_1 x_2 = 0$ est satisfaite à 10^{-4} près, ce qui ne suffit pas à être optimal.
- Avec une mise à jour dynamique, on converge vers l'optimum en 56 itérations avec l'heuristique 1 (31 avec l'heuristique 2).

Notons que cette règle de mise à jour dynamique n'a pas été testée sur d'autre problème pour l'instant et que ce sera donc l'objet de travaux futurs.

3.2.5 Problèmes de structure

Nous donnons ci-dessous les résultats numériques associés aux problèmes présentés dans le chapitre 1. Notons qu'à l'heure actuelle, il n'existe aucune collection de test pour les MPVC. Ainsi, nous avons créé tous les modèles MPVC, en langage AMPL, présentés dans cette section.

3.2.5.1 Minimisation du volume

Voyons comment s'est comporté PSUPERB sur les trois problèmes de structure dont on cherche à minimiser le volume.

Formulation Classique (1.50)

La valeur de l'objectif à l'optimalité est de 40.997. Les diamètres des barres sont

$$a_1 = 10, a_2 = 10, a_3 = 1, a_4 = 14.14 \text{ et } a_5 = 0.$$

Tableau 3.6: Résultats sur le problème à 5 barres avec formulation classique

	Heuristique 1	Heuristique 2
Version 1	échec	échec
Version 2	Optimal - 282 itérations	Optimal - 165 itérations
Version 3	échec	échec

Tableau 3.7: Résultats sur le problème à 10 barres avec formulation classique

	Heuristique 1	Heuristique 2
Version 1	échec	échec
Version 2	Optimal - 165 itérations	échec
Version 3	échec	échec

La valeur de l'objectif à l'optimalité est de 8553.44. Les diamètres des barres sont

$$\begin{aligned} a_1 &= 5 & a_2 &= 5 \\ a_3 &= 70.36 & a_4 &= 0 \\ a_5 &= 40.62 & a_6 &= 0 \\ a_7 &= 14.14 & a_8 &= 0 \\ a_9 &= 0 & a_{10} &= 68.32 \end{aligned}$$

Tableau 3.8: Résultats sur le problème à 18 barres avec formulation classique

	Heuristique 1	Heuristique 2
Version 1	échec	échec
Version 2	Optimal - 292 itérations	échec
Version 3	échec	échec

La valeur de l'objectif à l'optimalité est de 9924.4. Les diamètres des barres sont

$$\begin{aligned} a_1 &= 10 & a_2 &= 5 & a_3 &= 57.45 \\ a_4 &= 0 & a_5 &= 0 & a_6 &= 0 \\ a_7 &= 0 & a_8 &= 0 & a_9 &= 0 \\ a_{10} &= 0 & a_{11} &= 68.32 & a_{12} &= 68.32 \\ a_{13} &= 0 & a_{14} &= 0 & a_{15} &= 7.07 \\ a_{16} &= 0 & a_{17} &= 0 & a_{18} &= 0 \end{aligned}$$

Formulation MPVC (1.52)

Tableau 3.9: Résultats sur le problème à 5 barres avec formulation MPVC

	Heuristique 1	Heuristique 2
Version 1	échec	échec
Version 2	Optimal - 138 itérations	Optimal - 363 itérations
Version 3	échec	échec

La valeur de l'objectif à l'optimalité est de 35.746. Les diamètres des barres sont

$$a_1 = 2.25, a_2 = 2.25, a_3 = 6.75, a_4 = 6.36 \text{ et } a_5 = 10.96.$$

On remarque alors que la modélisation **MPVC** permet d'aboutir à une **meilleure solution**, en terme d'objectif, que le modèle classique. Ce problème admet donc au moins deux minima locaux mais nous n'avons pas cherché à savoir si, théoriquement, d'autres minima existaient.

Tableau 3.10: Résultats sur le problème à 10 barres avec formulation MPVC

	Heuristique 1	Heuristique 2
Version 1	échec	échec
Version 2	Optimal - 762 itérations	échec
Version 3	échec	échec

La valeur de l'objectif à l'optimalité est de 8553.44. Les diamètres des barres sont

$$\begin{aligned} a_1 &= 5 & a_2 &= 5 \\ a_3 &= 70.36 & a_4 &= 0 \\ a_5 &= 40.62 & a_6 &= 0 \\ a_7 &= 14.14 & a_8 &= 0 \\ a_9 &= 0 & a_{10} &= 68.32 \end{aligned}$$

Tableau 3.11: Résultats sur le problème à 18 barres avec formulation MPVC

	Heuristique 1	Heuristique 2
Version 1	échec	échec
Version 2	Optimal - 645 itérations	échec
Version 3	échec	échec

La valeur de l'objectif à l'optimalité est de 9924.4. Les diamètres des barres sont

$$\begin{array}{lll} a_1 = 10 & a_2 = 5 & a_3 = 57.45 \\ a_4 = 0 & a_5 = 0 & a_6 = 0 \\ a_7 = 0 & a_8 = 0 & a_9 = 0 \\ a_{10} = 0 & a_{11} = 68.32 & a_{12} = 68.32 \\ a_{13} = 0 & a_{14} = 0 & a_{15} = 7.07 \\ a_{16} = 0 & a_{17} = 0 & a_{18} = 0 \end{array}$$

Étude comparative :

Comparons, dans le tableau 3.12, les résultats obtenus avec PSUPERB et ceux issus de tests réalisés avec le solveur commercial SNOPT. Le tableau 3.12 nous permet d'énoncer les remarques suivantes :

- PSUPERB est globalement moins efficace en termes de nombre d'itérations.
- Les solutions trouvées sont les mêmes en terme d'objectif à l'exception du problème à 5 barres formulé comme un MPVC pour lequel PSUPERB trouve une solution ayant un meilleur objectif.
- La formulation MPVC semble plus difficile à résoudre.

Cependant, notons que :

- Le code de PSUPERB n'est pas optimisé ; dans cette maîtrise on l'a simplement fait fonctionner et donc on peut conclure qu'il est prometteur.
- SNOPT a une dizaine d'années d'expérience.
- PSUPERB est fondé sur des résultats théoriques en ce qui concerne les MPVCs en particulier et plus généralement, les problèmes dégénérés.

On donne à présent l'allure physique des structures optimales à la figure 3.10.

Tableau 3.12: Comparaison des «Nombre d'itérations / Objectifs » des solveurs PSUPERB et SNOPT sur les problèmes de minimisation de volume de structure

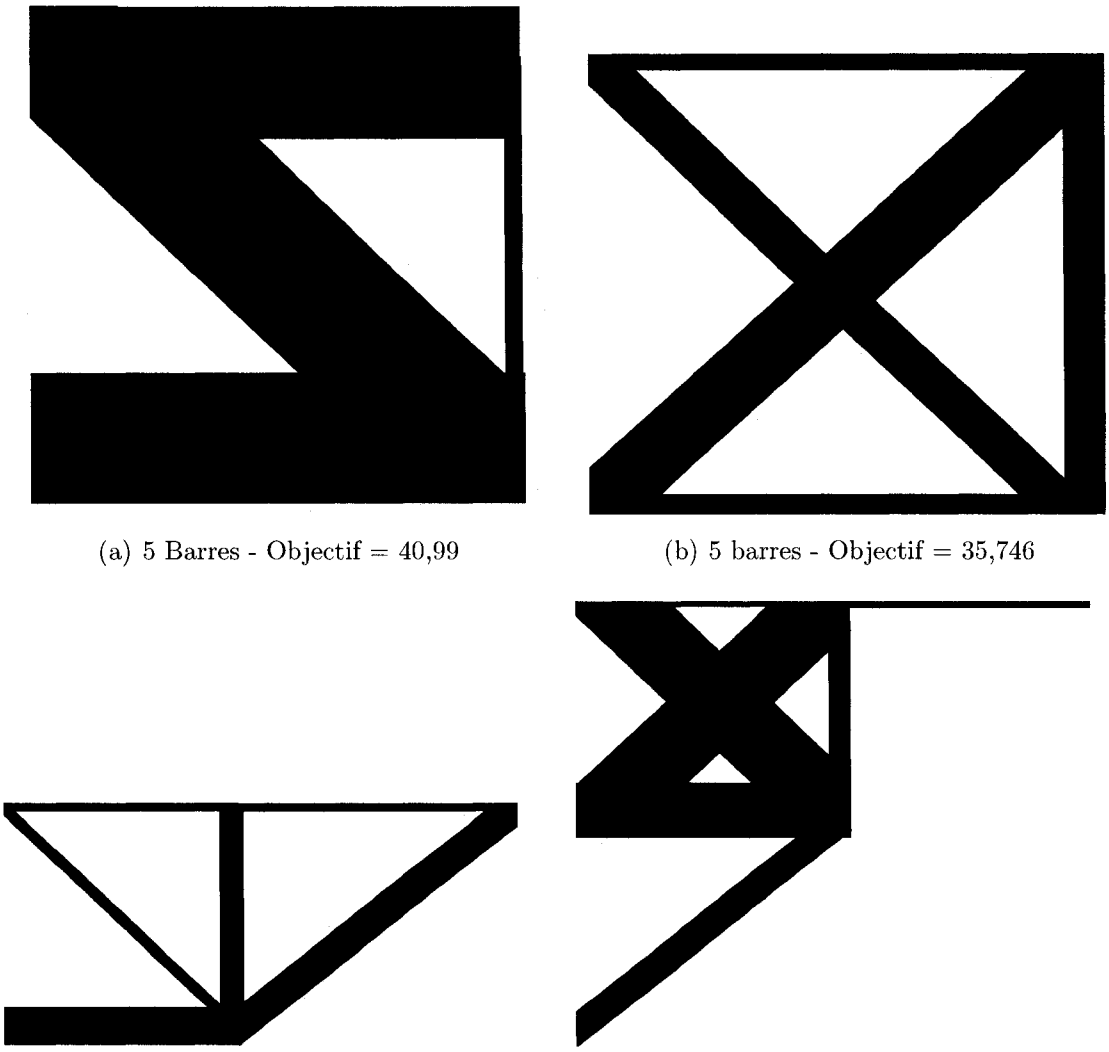
		PSUPERB	SNOPT
Classique	5 barres	165 / 40,99	26 / 40,99
	10 barres	165 / 8553,4	39 / 8553,4
	18 barres	292 / 9924,4	76 / 9924,4
MPVC	5 barres	138 / 35,746	18 / 40,99
	10 barres	508 / 8553,4	52 / 8553,4
	18 barres	645 / 9924,4	80 / 9924,4

Remarque : Dans toutes les solutions trouvées, à l'exception de la solution à 5 barres dont l'objectif vaut 35,746, il y a une barre dont le diamètre est nul donc la LICQ n'est pas satisfaite pour ces structures optimales d'après la propriété 1.3.1.1 du chapitre 1.

Conclusion : Sur les problèmes de structure, la Version 2 est indispensable. Les autres échouent toujours, même sur le problème le plus simple. Concernant les deux modélisations, classique et MPVC, la formulation MPVC semble plus complexe à résoudre même si elle peut amener à trouver un meilleur objectif. En terme de valeur objectif, PSUPERB semble très compétitif par rapport à un solveur commercial tel que SNOPT (Gill et al., 2005) en dépit d'un nombre plus élevé d'itérations pour atteindre cet objectif.

3.2.5.2 Minimsation de l'énergie de déformation

On considère ici les deux formulations, (1.53) et (1.56), du chapitre 1. En observant les résultats obtenus par les diverses versions de PSUPERB on a travaillé, pour ces problèmes, avec la Version 2. Pour la première formulation, on obtient alors le tableau 3.13. Si on s'intéresse au second problème, (1.56), qui est un MPEC, on obtient le tableau 3.14. De toute évidence, les résultats ne sont pas brillants. Soulignons toutefois que parmi les solveurs disponibles sur la marché, seul SNOPT parvient à trouver des points optimaux pour les premiers problèmes ; pour les seconds, dès qu'on dépasse les tailles académiques, l'échec se généralise à tous les solveurs (voir les détails dans (Kocvara & Outrata, 2006)).



(a) 5 Barres - Objectif = 40,99

(b) 5 barres - Objectif = 35,746

(c) 10 barres - Objectif = 8553,4

(d) 18 barres - Objectif = 9924,4

Figure 3.10 – Structures optimales

Tableau 3.13: Nombre d'itérations effectuées par la Version 2 de PSUPERB sur des problèmes de type (1.53)

	Heuristique 1	Heuristique 2
Treillis 3x3	825	échec
Treillis 4x4	2140	échec
Treillis 5x5	3247	échec
Treillis 6x2	échec	échec
Treillis 41x9	échec	échec

Conclusion : On retiendra que les problèmes d'optimisation de structure, formulés comme des MPEC, sont des problèmes encore bien peu maîtrisés numériquement.

Tableau 3.14: Nombre d'itérations effectuées par PSUPERB sur des problèmes de type (1.56)

	Heuristique 1	Heuristique 2
Treillis 3x3	échec	3900
Treillis 4x4	échec	échec
Treillis 5x5	échec	échec

3.3 Visualisation

Puisque nous nous sommes concentrés sur des problèmes de structure, une ambition légitime est de représenter l'évolution du treillis de barres au cours des itérations. Pour cela nous avons utilisé une application graphique appelée **Processing**, disponible gratuitement sur `www.processing.org`. Le but de cette application est de proposer un environnement graphique utilisable seul ou en interaction avec d'autres applications. Le langage sous-jacent à cette application est `Java`.

Dans notre cas, nous avons créé un lien entre Processing et l'implémentation de PSUPERB. La passerelle entre les deux applications est un fichier texte, `valeur.txt`, dans lequel `psuperb.py` écrit et qui est ensuite interprété par Processing. Concrètement voici ce qui se passe : `valeur.txt` contient les diamètres des barres du treillis. Processing lit `valeur.txt` toutes les 0.02 secondes et affiche le dessin qui correspond. Lorsqu'on fait tourner `psuperb.py`, à chaque fin d'itération externe, on écrit dans `valeur.txt` les valeurs courantes des variables, c'est-à-dire des diamètres des barres. Au final, on a l'impression de visualiser l'évolution en temps réel du treillis proposé par l'algorithme.

Un aperçu de ce qui apparaît à l'écran est donné aux figures 3.11 et 3.12.

On voit alors que certains traits sont plus épais que d'autres : cela correspond aux différences entre les diamètres pour chacune des barres. Le code nécessaire à

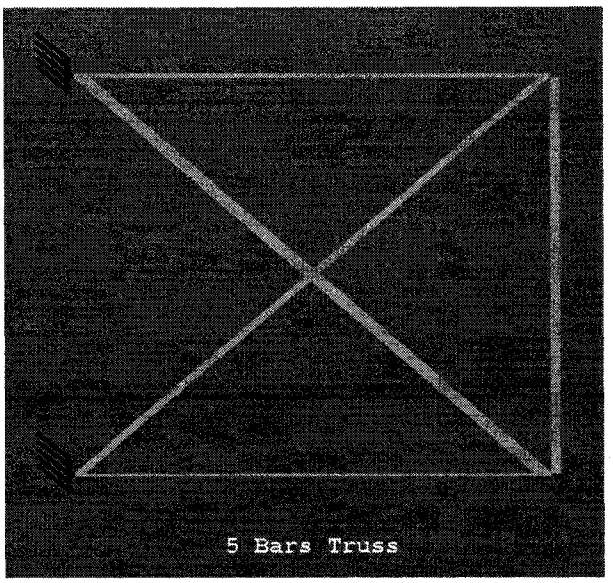


Figure 3.11 – Aperçu de la visualisation du treillis 5 barres proposée par Processing

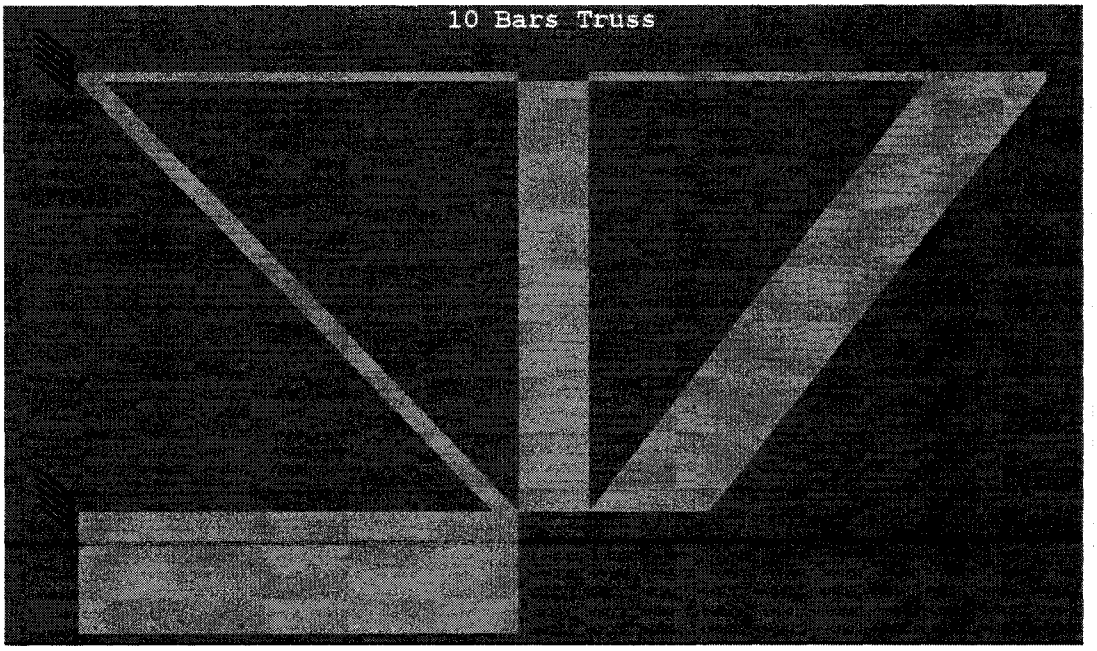


Figure 3.12 – Aperçu de la visualisation du treillis 10 barres proposée par Processing

l'obtention de ces images est disponible à l'adresse suivante :

`perso.telecom-paristech.fr/~curatolo/Piero/Work/Work.html`.

Pour le moment, nous avons écrit le code pour les problèmes des treillis à 5 et 10 barres uniquement. En fait, nous pourrions le faire pour un treillis quelconque, du moment qu'on connait les positions horizontale et verticale de toutes les barres (afin de faire correspondre les diamètres avec les bonnes barres). Lorsque le nombre de barres devient grand, cela peut s'avérer fastidieux. Il serait alors possible d'imaginer d'autres angles de vue, plus simple à représenter. Dans notre cas, nous voulions simplement proposer une première illustration des solutions calculées par notre algorithme. Nous l'avons donc fait sur des exemples simples afin de vérifier, par le bon sens, que les solutions proposées avaient l'air physiquement réalisables !

Remarque : la figure 3.12 est conforme à la solution proposée dans (Stolpe & Svanberg, 2003).

3.4 Difficultés et travaux futurs

3.4.1 Problèmes non-résolus

Lors des tests sur la collection de problèmes (Hock & Schittkowski, 1981), PSUPERB n'a parfois pas été capable de trouver de solution optimale. Certains problèmes se sont avérés résolus par un bon choix (différent de ceux proposés à la section 2.2.1) du paramètre de pénalisation initial. D'autres ont continué à mettre en défaut l'algorithme. Ci-dessous nous dressons une liste des divers problèmes non-résolus avec dans certains cas la valeur du ν initial qui a permis d'arriver à l'optimalité.

Problèmes résolus :
HS088, 89, 90, 91, 92 avec $\nu_0 = (600,3000,600)$
HS067 avec la Version 2 et $\nu_0 = (100,150,100)$
HS013 : Voir chapitre 3 section 3.1 pour une explication du comportement de PSUPERB sur ce problème.

Problèmes non-résolus :
HS072, 74, 75, 99 : Ces problèmes, non résolus par PSUPERB, sont pourtant assez facilement résolus par SNOPT (moins de 50 itérations).
HS087 : Ce problème n'est pas différentiable.

3.4.2 Difficultés

3.4.2.1 Le paramètre ν

Comme nous l'avons vu jusqu'ici, le paramètre de pénalisation, ν, est déterminant dans la convergence de l'algorithme vers une solutions optimale ou non. Deux difficultés interviennent lorsqu'on étudie ce paramètre : le choix initial et la mise à jour. En effet, en dépit des heuristiques proposées, on a vu que certains problèmes pouvaient être résolus par un choix judicieux du ν initial alors que les heuristiques échouaient à trouver une solution optimale. Ensuite, concernant la règle de mise à jour, on a privilégié une implémentation qui semblait fonctionner dans de nombreux cas mais on peut se demander si une autre règle de mise à jour ne pourrait pas permettre de résoudre certains problèmes encore sources d'échec ou bien améliorer les résultats actuels.

3.4.2.2 Étude de l'influence de μ

Lors de ces travaux, nous avons beaucoup insisté sur l'influence du paramètre ν mais peu évoqué le rôle de μ. Il y a fort à parier que le choix initial pour ce paramètre influence les résultats ainsi que sa règle de mise à jour (voir section 2.1.4.3 pour les règles adoptées jusqu'ici). C'est pourquoi nous souhaitons ici mentionner cette piste de recherche afin d'ouvrir la voie à de futurs travaux à ce sujet.

3.4.2.3 Choix des multiplicateurs primaux-duaux initiaux

Dans notre étude, nous avons choisi de prendre comme valeur initiale des estimateurs primaux-duaux celle des multiplicateurs primaux. Évidemment, ce choix n'a aucune raison d'être meilleur qu'un autre. Notons que personne aujourd'hui ne sait s'il existe un choix optimal à effectuer. Il fallait donc opter parmi les multiples options possibles. Nous laissons ainsi la porte ouverte à de futures recherches dans l'optique de déterminer un jeu d'estimateurs initiaux meilleurs que ceux que nous avons adopté.

Conclusion

Nous avons essayé de résoudre le problème de conception de structure. Ce problème se modélise suivant différents paradigmes de programmation mathématique. Nous avons choisi une méthode de pénalisation adaptée à la résolution de tels problèmes. Cette méthode, baptisée PSUPERB, est un algorithme de points intérieurs avec pénalisation ℓ_1 et variables élastiques.

Dans un premier temps, nous avons cherché à découvrir les spécificités de chaque formulation, notamment en termes de satisfaction des conditions de qualification. Ainsi, nous avons pu constater que l'algorithme sélectionné était tout particulièrement adapté à la situation grâce à l'emploi des variables élastiques. Nous sommes alors passés à l'implémentation de la méthode en code Python afin de pouvoir résoudre numériquement ces problèmes.

Théoriquement, l'algorithme est en mesure de trouver des points optimaux qui vérifient les conditions de KKT ou bien de prouver que des conditions de qualification (MFCQ ou VC-MFCQ selon ce qui s'applique) sont violées au point de convergence. Cette capacité à décerner un certificat de dégénérescence est peu répandue parmi les codes actuellement disponibles. Numériquement, les résultats sont très satisfaisants sur les collections de problèmes Hock-Schittkowski et MacMPEC ainsi que sur des problèmes de structure de petite taille.

Toutefois, les performances actuelles de l'algorithme peuvent encore être fortement optimisées. Il nous reste par exemple à trouver comment mieux gérer les paramètres de pénalisation et barrière ou encore à raffiner les stratégies utilisées lors de la résolution des sous-problèmes de région de confiance. Nous devrons également traiter des problèmes de structure de grande taille ou d'autres types de problèmes dégénérés afin de mieux évaluer l'étendue des capacités de PSUPERB. Enfin, en rapport avec l'implémentation, il reste de nombreuses opportunités à saisir pour rendre le code plus compétitif.

Bibliographie

Achtziger, W.(1999). Local stability of trusses in the context of topology optimization. *Structural Optimization, 17*(4), 235-246.

Achtziger, W., & Kanzow, C. (2008). Mathematical programs with vanishing constraints : optimality conditions and constraints qualifications. *Mathematical Programming, 114*(1), 69-99.

Byrd, R. H., Gilbert, J.-C., & Nocedal, J.(2000a). A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming, Series A, 89*(1), 149-185.

Byrd, R. H., Gilbert, J.-C., & Nocedal, J.(2000b). A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming A, 89*(1), 149–185.

Byrd, R. H., Hribar, M. E., & Nocedal, J.(1999a). An interior point method for large scale nonlinear programming. *SIAM Journal on Optimization, 9*, 877-900.

Byrd, R. H., Hribar, M. E., & Nocedal, J.(1999b). An interior point method for large scale nonlinear programming. *9*, 877–900.

Conn, A. R., Gould, N. I. M., Orban, D., & Toint, P. L. (2000). A primal-dual trust-region algorithm for non-convex nonlinear optimization. *Mathematical Programming, 87*(2), 215-249.

Conn, A. R., Gould, N. I. M., & Toint, P. L.(2000). *Trust-Region Methods.* Philadelphia, USA : SIAM.

Coulibaly, Z. (2008). *Une méthode élastique pour le traitement des programmes mathématiques avec contraintes d'équilibre.* (En préparation)

Dolan, E., & Moré, J. J. (2001). Benchmarking optimization software with performance profiles. *Mathematical Programming, 91*(2), 201-213.

Fiacco, A. V., & McCormick, G. P. (1968). *Nonlinear programming : Sequential unconstrained minimization techniques.* New-York, NY : John Wiley & Sons.

Forsgren, A., Gill, P. E., & Wright, M. H. (2002). Interior Methods for Nonlinear Optimization. *SIAM Review, 44*(4), 525-597.

Fourer, R., Gay, D., & Kernighan, B.(2003). *AMPL, a modeling language for mathematical programming* (2nd ed.). Duxbury.

Gauvin, J. (1977). A necessary and sufficient regularity condition to have bounded multipliers in nonconvex programming. *Mathematical Programming, 12*(1), 136-138.

Gill, P. E., Murray, W., & Saunders, M. A.(2005). SNOPT : An SQP Algorithm for Large-Scale Constrained Optimization. *SIAM Review, 47*(1), 99-131.

Gould, N. I. M. (1985). On practical conditions for the existence and uniqueness of solutions to the general equality quadratic-programming problem. *Mathematical Programming, 32*(1), 90-99.

Gould, N. I. M., Orban, D., & Toint, P. L.(2003). *An interior-point ℓ1-penalty method for nonlinear optimization* (Technical Report RAL-TR-2003-022). Oxfordshire OX11 OQX : Computational Science and Engineering Departement, Rutherford Appleton Laboratory.

Guo, X., Cheng, G., & Yamazaki, K. (2001). A new approach for the solution of singular optima in truss topology optimization with stress and local buckling constraints. *Structural Multidisciplinary Optimization, 22*(5), 364-372.

Hock, W., & Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes.* Springer-Verlag.

Kocvara, M., & Outrata, J. (2006). Effective reformulations of the truss topology design problem. *Optimization and Engineering, 7*(2), 201-219.

Leyffer, S. (2005). *MacMPEC AMPL collection of Mathematical Programs with Equilibrium Constraints.* (`www-unix.mcs.anl.gov/~leyffer/MacMPEC/`)

Leyffer, S., López-Calva, G., & Nocedal, J.(2006). Interior points method for mathematical programs with complementarity constraints. *SIAM Journal on Optimization, 17*(1), 52-77.

Mangasarian, O. L.(1994). *Nonlinear Programming.* SIAM.

Mangasarian, O. L., & Fromowitz, S. (1967). The Fritz-John necessary optimality conditions in the presence of equality and inequality constraints. *Journal of Mathematical Analysis and Applications, 17*, 37-47.

McCormick, G. P. (1967). Second order conditions for constrained minima. *SIAM Journal on Applied Mathematics, 15*, 641-652.

Murtagh, B. A., & Saunders, M. A.(1998). *MINOS 5.5 User's Guide* (Report SOL 83-20R). Dept of Operations Research, Stanford University.

Nocedal, J., & Yuan, Y.(1998). Combining trust region and line search techniques. In Y. Yuan (Ed.), *Advances in Nonlinear Programming* (pp. 153–175). Kluwer.

Orban, D., & Friedlander, M. P. (2003). *NLPy : A Nonlinear Programming Package in Python.* (`nlpy.sourceforge.net`)

Scheel, H., & Scholtes, S. (2000). Mathematical programs with complementarity constraints : Stationarity, optimality and sensitivity. *Mathematics of Operations Research, 25*(1), 1-22.

Stolpe, M., & Svanberg, K. (2003). A note on stress-constrained truss toplogy optimization. *Structural Multidisciplinary Optimization, 25*(1), 62-64.

Vanderbei, R. J., & Shanno, D. F. (1999). An interior point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications, 13*, 231-252.

Wright, M. H. (1992). Interior Methods for Constrained Optimization. *Acta Numerica, 1*, 341-407.